# Bài thực hành bổ sung

## A. Program for FCFS CPU Scheduling | Set 1

Last Updated : 21 Jul, 2023

Nguồn: https://www.geeksforgeeks.org/program-for-priority-cpu-scheduling-set-1/?ref=ml_lbp

Given n processes with their burst times, the task is to find average waiting time and average turn around time using FCFS scheduling algorithm.

First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.

In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

Here we are considering that arrival time for all processes is 0.

How to compute below times in Round Robin using a program?

**Completion Time:** Time at which process completes its execution.

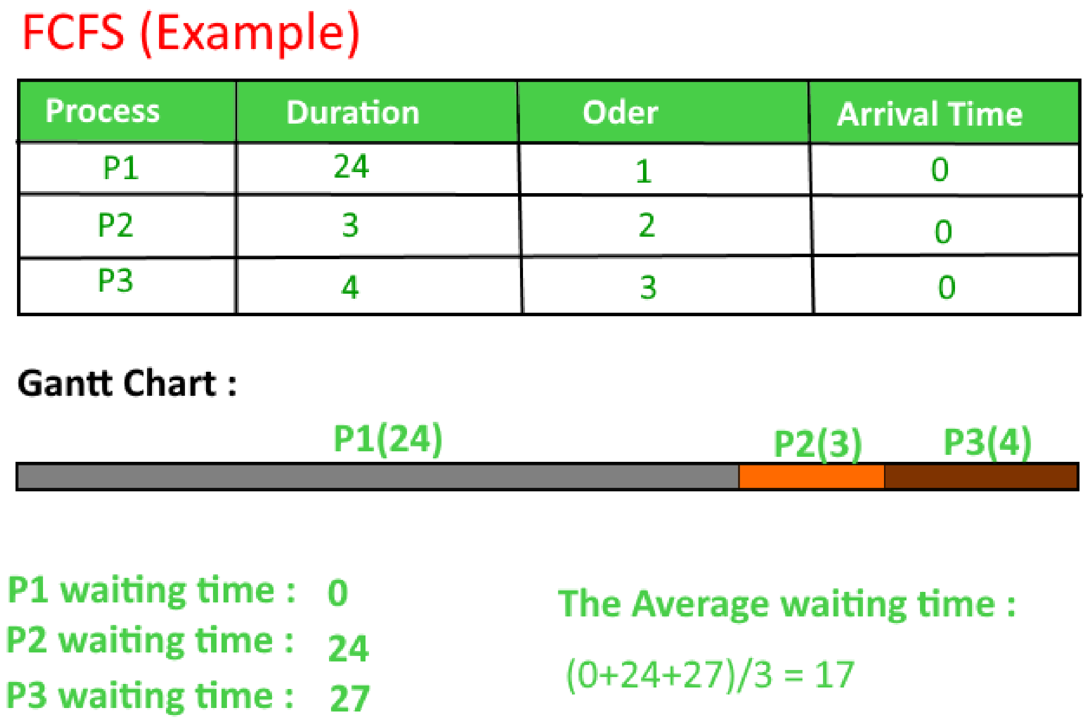
**Turn Around Time:** Time Difference between completion time and arrival time.

**Turn Around Time** = Completion Time – Arrival Time

**Waiting Time(W.T):** Time Difference between turn around time and burst time.

**Waiting Time** = Turn Around Time – Burst Time

In this post, we have assumed arrival times as 0, so turn around and completion times are same.



FCFS (Example)

| Process | Duration | Oder | Arrival Time |
|---|---|---|---|
| P1 | 24 | 1 | 0 |
| P2 | 3 | 2 | 0 |
| P3 | 4 | 3 | 0 |

Gantt Chart :

P1 waiting time : 0
P2 waiting time : 24
P3 waiting time : 27

The Average waiting time :

(0+24+27)/3 = 17

**Implementation:**

1- Input the processes along with their burst time (bt).

2- Find waiting time (wt) for all processes.

3- As first process that comes need not to wait so

waiting time for process 1 will be 0 i.e. wt[0] = 0.

4- Find **waiting time** for all other processes i.e. for process i ->
wt[i] = bt[i-1] + wt[i-1] .

5- Find **turnaround time** = waiting_time + burst_time for all processes.

6- Find **average waiting time** =
total_waiting_time / no_of_processes.

7- Similarly, find **average turnaround time** =
total_turn_around_time / no_of_processes.

**C++**

```
// C++ program for implementation of FCFS
// scheduling
#include<iostream>
using namespace std;
// Function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
        // waiting time for first process is 0
        wt[0] = 0;
        // calculating waiting time
        for (int i = 1; i < n ; i++ )
                wt[i] = bt[i-1] + wt[i-1] ;
}
// Function to calculate turn around time
void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[])
{
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}

//Function to calculate average time
void findavgTime( int processes[], int n, int bt[])
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;
```

```
    //Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt);
    //Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);
    //Display processes along with all details
    cout << "Processes "<< " Burst time "
        << " Waiting time " << " Turn around time\n";
    // Calculate total waiting time and total turn
    // around time
    for (int i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
            << wt[i] <<"\t\t " << tat[i] <<endl;
    }

    cout << "Average waiting time = "<< (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "  << (float)total_tat / (float)n;
}

// Driver code
int main()
{
    //process id's
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];
    //Burst time of all processes
    int burst_time[] = {10, 5, 8};
```

```
        findavgTime(processes, n, burst_time);
        return 0;
}
```

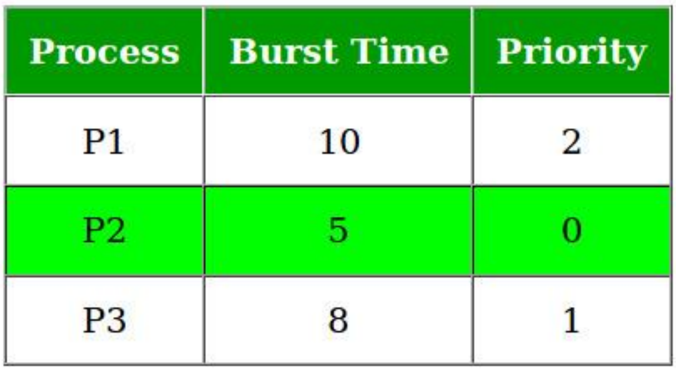
## B. Program for Priority CPU Scheduling | Set 1

Nguồn: https://www.geeksforgeeks.org/program-for-priority-cpu-scheduling-set-1/?ref=ml_lbp

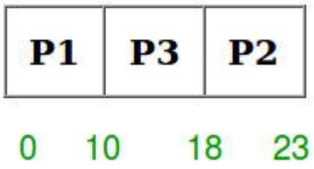
Priority scheduling is one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. The process with the highest priority is to be executed first and so on. Processes with the same priority are executed on a first-come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement. Also priority can be decided on the ratio of average I/O to average CPU burst time.

**Implementation:**

```
1- First input the processes with their burst time
   and priority.
2- Sort the processes, burst time and priority
   according to the priority.
3- Now simply apply FCFS algorithm.
```



| Process | Burst Time | Priority |
|---|---|---|
| P1 | 10 | 2 |
| P2 | 5 | 0 |
| P3 | 8 | 1 |



```
// C++ program for implementation of FCFS
// scheduling
#include <bits/stdc++.h>
using namespace std;
struct Process {
        int pid; // Process ID
        int bt; // CPU Burst time required
        int priority; // Priority of this process
};

// Function to sort the Process acc. to priority
bool comparison(Process a, Process b)
{
        return (a.priority > b.priority);
}

// Function to find the waiting time for all
// processes
void findWaitingTime(Process proc[], int n, int wt[])
{
        // waiting time for first process is 0
        wt[0] = 0;
        // calculating waiting time
        for (int i = 1; i < n; i++)
                wt[i] = proc[i - 1].bt + wt[i - 1];
}
// Function to calculate turn around time
void findTurnAroundTime(Process proc[], int n, int wt[], int tat[])
{
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n; i++)
                tat[i] = proc[i].bt + wt[i];
}

// Function to calculate average time
void findavgTime(Process proc[], int n)
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;
        // Function to find waiting time of all processes
        findWaitingTime(proc, n, wt);
        // Function to find turn around time for all processes
        findTurnAroundTime(proc, n, wt, tat);
        // Display processes along with all details
        cout << "\nProcesses " << " Burst time "  << " Waiting time "  << " Turn around time\n";
        // Calculate total waiting time and total turn
        // around time
        for (int i = 0; i < n; i++) {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                cout << " " << proc[i].pid << "\t\t" << proc[i].bt
                    << "\t " << wt[i] << "\t\t " << tat[i]
                    << endl;
        }
        cout << "\nAverage waiting time = " << (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "  << (float)total_tat / (float)n;
}
```

```
void priorityScheduling(Process proc[], int n)
{
        // Sort processes by priority
        sort(proc, proc + n, comparison);
        cout << "Order in which processes gets executed \n";
        for (int i = 0; i < n; i++)
                cout << proc[i].pid << " ";
        findavgTime(proc, n);
}

// Driver code
int main()
{
        Process proc[]      = { { 1, 10, 2 }, { 2, 5, 0 }, { 3, 8, 1 } };
        int n = sizeof proc / sizeof proc[0];
        priorityScheduling(proc, n);
        return 0;
}
```

**Output:**

```
Order in which processes gets executed
1 3 2
Processes  Burst time  Waiting time  Turn around time
 1        10     0         10
 3        8     10         18
 2        5     18         23

Average waiting time = 9.33333
Average turn around time = 17
```

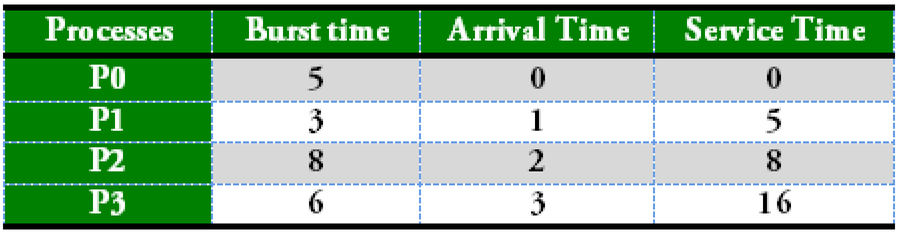
### C. Program for FCFS CPU Scheduling | Set 2 (Processes with different arrival times)

Nguồn: https://www.geeksforgeeks.org/program-for-fcfs-cpu-scheduling-set-2-processes-with-different-arrival-times/?ref=ml_lbp

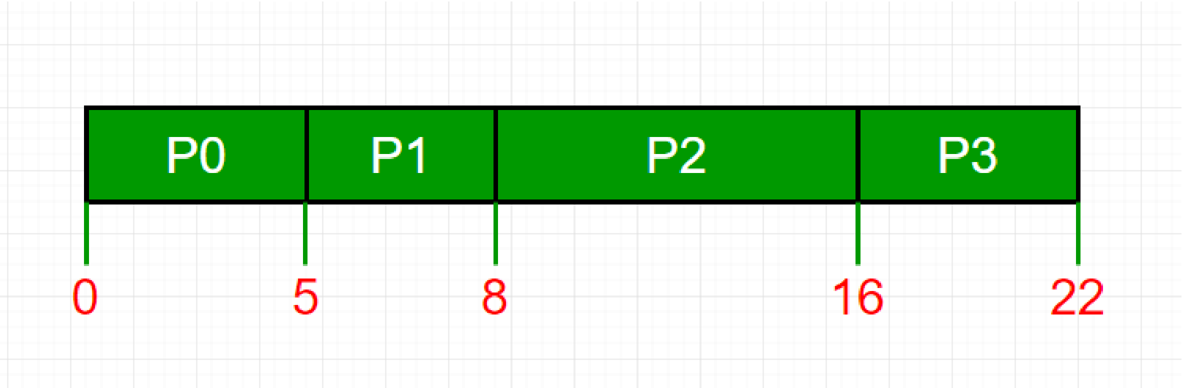
Last Updated : 13 Sep, 2023

We have already discussed FCFS Scheduling of processes with same arrival time.
In this post, scenarios, when processes have different arrival times, are discussed. Given n processes with their burst times and arrival times, the task is to find the average waiting time and an average turn around time using FCFS scheduling algorithm.
FIFO simply queues processes in the order they arrive in the ready queue. Here, the process that comes first will be executed first and next process will start only after the previous gets fully executed.

1. Completion Time: Time at which the process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time.
   Turn Around Time = Completion Time – Arrival Time
3. Waiting Time(W.T): Time Difference between turn around time and burst time.
   Waiting Time = Turn Around Time – Burst Time.



| Processes | Burst time | Arrival Time | Service Time |
|---|---|---|---|
| P0 | 5 | 0 | 0 |
| P1 | 3 | 1 | 5 |
| P2 | 8 | 2 | 8 |
| P3 | 6 | 3 | 16 |



```
Process     Wait Time : Service Time - Arrival Time
   P0                    0 - 0    = 0
   P1                    5 - 1    = 4
   P2                    8 - 2    = 6
   P3                    16 - 3  = 13

Average Wait Time: (0 + 4 + 6 + 13) / 4 = 5.75
```

**Service Time:** Also known as Burst Time, this is the amount of time a process requires to complete its execution on the CPU. It represents the time the CPU spends executing instructions of that particular process.

**Waiting Time:** It refers to the total amount of time that a process spends waiting in the ready queue before it gets a chance to execute on the CPU.

***Changes in code as compare to code of FCFS with same arrival time:*** *To find waiting time: Time taken by all processes before the current process to be started (i.e. burst time of all previous processes) – arrival time of current process*

***wait_time[i] = (bt[0] + bt[1] +…… bt[i-1] ) – arrival_time[i]***

**Implementation:**

```
1- Input the processes along with their burst time(bt)
   and arrival time(at)
2- Find waiting time for all other processes i.e. for
   a given process  i:
       wt[i] = (bt[0] + bt[1] +...... bt[i-1]) - at[i]
3- Now find turn around time
          = waiting_time + burst_time for all processes
4- Average waiting time =
                    total_waiting_time / no_of_processes
5- Average turn around time =
                 total_turn_around_time / no_of_processes
```

**Example1:**

```
//Process Synced, dynamic input, c++ easy to understand code

#include<iostream>

#include<stdlib.h>

using namespace std;

//class process with all the time values and functions

class Process{

        int id, bt, at, ct, tat, wt;

        public:

        void input(Process*,int );

        void calc(Process *,int);

        void show(Process*,int);

        void sort(Process *, int);

};
```

```
//main function
int main(){
    int n;
    cout<<"\nEnter the no of processes in your system:\n";
    cin>>n;
    Process *p = new Process[n];
    Process f;
    f.input(p,n);
    f.sort(p, n);
    f.calc(p,n);
    f.show(p,n);
    return 0;
}
//taking input arrival and burst times for all processes
void Process::input(Process *p,int n){
    for(int i = 0;i<n;i++){
        cout<<"\nEnter pival time for process "<<i+1<<":\n";
        cin>>p[i].at;
        cout<<"\nEnter burst time for process "<<i+1<<":\n";
        cin>>p[i].bt;
        p[i].id = i+1;
    }
}
//calculating waiting, turn-around and completion time
void Process::calc(Process*p, int n){
    int sum = 0;
    sum = sum + p[0].at;
    for(int i = 0;i<n;i++){
        sum = sum + p[i].bt;
        p[i].ct = sum;
        p[i].tat = p[i].ct - p[i].at;
        p[i].wt = p[i].tat - p[i].bt;
        if(sum<p[i+1].at){
            int t = p[i+1].at-sum;
            sum = sum+t;
        }
    }
}
//Sorting processes with respect to arrival times (needed for synchronized input)
void Process::sort(Process*p, int n){
    for(int i=0;i<n-1;i++){
        for(int j=0;j<n-i-1;j++){
            if(p[j].at>p[j+1].at){
                int temp;
                //sorting burst times
                temp = p[j].bt;
                p[j].bt = p[j+1].bt;
                p[j+1].bt = temp;
                //sorting arrival times
                temp = p[j].at;
                p[j].at = p[j+1].at;
                p[j+1].at = temp;
                //sorting their respective IDs
                temp = p[j].id;
                p[j].id = p[j+1].id;
                p[j+1].id = temp;
            }
        }
        }
}
//display function
void Process::show(Process*p, int n){
        cout<<"Process\tArrival\tBurst\tWaiting\tTurn Around\tCompletion\n";
        for(int i =0;i<n;i++){
                cout<<" P["<<p[i].id<<"]\t "<<p[i].at<<"\t"<<p[i].bt<<"\t"<<p[i].wt<<"\t "<<p[i].tat<<"\t\t"<<p[i].ct<<"\n";
        }
}
```

**Output:**

```
Processes  Burst Time  Arrival Time  Waiting Time  Turn-Around Time  Completion Time
 1       5       0       0       5        5
 2       9       3       2       11       14
 3       6       6       8       14       20
Average waiting time = 3.33333
Average turn around time = 10.0
```

## D. Priority CPU Scheduling with different arrival time – Set 2

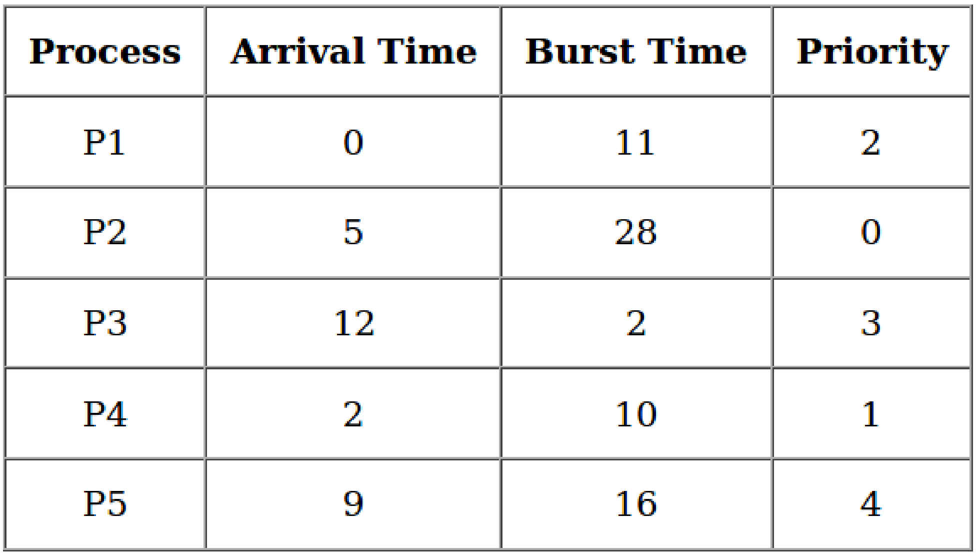
**Nguồn:** https://www.geeksforgeeks.org/priority-cpu-scheduling-with-different-arrival-time-set-2/?ref=ml_lbp

Last Updated : 17 Apr, 2024

**Prerequisite –** Program for Priority Scheduling – Set 1
Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned first arrival time (less arrival time process first) if two processes have same arrival time, then compare to priorities (highest process first). Also, if two processes have same priority then compare to process number (less process number first). This process is repeated while all process get executed.
**Implementation –**

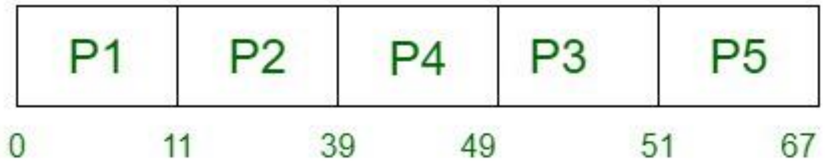
1. First input the processes with their arrival time, burst time and priority.

2. First process will schedule, which have the lowest arrival time, if two or more processes will have lowest arrival time, then whoever has higher priority will schedule first.
3. Now further processes will be schedule according to the arrival time and priority of the process. (Here we are assuming that lower the priority number having higher priority). If two process priority are same then sort according to process number. **Note:** In the question, They will clearly mention, which number will have higher priority and which number will have lower priority.
4. Once all the processes have been arrived, we can schedule them based on their priority.



| Process | Arrival Time | Burst Time | Priority |
|---|---|---|---|
| P1 | 0 | 11 | 2 |
| P2 | 5 | 28 | 0 |
| P3 | 12 | 2 | 3 |
| P4 | 2 | 10 | 1 |
| P5 | 9 | 16 | 4 |

**Gantt Chart –**



**Examples –**

**Input :**

```
process no-> 1 2 3 4 5
arrival time-> 0 1 3 2 4
burst time-> 3 6 1 2 4
priority-> 3 4 9 7 8
```

**Output :**

```
Process_no  arrival_time  Burst_time  Complete_time  Turn_Around_Time    Waiting_Time
1           0             3           3              3            0
2           1             6           9              8            2
3           3             1           16              13           12
4           2             2           11               9            7
5           4             4           15               11           7
```

```
Average Waiting Time is : 5.6
Average Turn Around time is : 8.8
```

```
// C++ implementation for Priority Scheduling with
//Different Arrival Time priority scheduling
/*1. sort the processes according to arrival time
2. if arrival time is same the acc to priority
3. apply fcfs
*/
#include <bits/stdc++.h>
using namespace std;
#define totalprocess 5
// Making a struct to hold the given input
struct process
{
    int at,bt,pr,pno;
};

process proc[50];

/*
Writing comparator function to sort according to priority if arrival time is same
*/
bool comp(process a,process b)
{
    if(a.at == b.at)
    {
        return a.pr<b.pr;
    }
    else
```

```
        {
           return a.at<b.at;
        }
}

// Using FCFS Algorithm to find Waiting time
void get_wt_time(int wt[])
{
        // declaring service array that stores cumulative burst time
        int service[50];
        // Initialising initial elements of the arrays
        service[0] = proc[0].at;
        wt[0]=0;
        for(int i=1;i<totalprocess;i++)
        {
                service[i]=proc[i-1].bt+service[i-1];
                wt[i]=service[i]-proc[i].at;
                // If waiting time is negative, change it into zero
                   if(wt[i]<0)
                   {
                        wt[i]=0;
                   }
        }

}

void get_tat_time(int tat[],int wt[])
{
        // Filling turnaroundtime array
        for(int i=0;i<totalprocess;i++)
        {
                tat[i]=proc[i].bt+wt[i];
        }
}

void findgc()
{
        //Declare waiting time and turnaround time array
        int wt[50],tat[50];
        double wavg=0,tavg=0;
        // Function call to find waiting time array
        get_wt_time(wt);
        //Function call to find turnaround time
        get_tat_time(tat,wt);
        int stime[50],ctime[50];
        stime[0] = proc[0].at;
        ctime[0]=stime[0]+tat[0];
        // calculating starting and ending time
        for(int i=1;i<totalprocess;i++)
           {
              stime[i]=ctime[i-1];
              ctime[i]=stime[i]+tat[i]-wt[i];
           }

        cout<<"Process_no\tStart_time\tComplete_time\tTurn_Around_Time\tWaiting_Time"
        <<endl;
          // display the process details
         for(int i=0;i<totalprocess;i++)
```

```
            {
              wavg += wt[i];
              tavg += tat[i];
                cout<<proc[i].pno<<"\t\t"<<
                  stime[i]<<"\t\t"<<ctime[i]<<"\t\t"<<
                  tat[i]<<"\t\t\t"<<wt[i]<<endl;
            }

              // display the average waiting time
              //and average turn around time

            cout<<"Average waiting time is : ";
            cout<<wavg/(float)totalprocess<<endl;
            cout<<"average turnaround time : ";
            cout<<tavg/(float)totalprocess<<endl;
}

int main()
{
        int arrivaltime[] = { 1, 2, 3, 4, 5 };
        int bursttime[] = { 3, 5, 1, 7, 4 };
        int priority[] = { 3, 4, 1, 7, 8 };

        for(int i=0;i<totalprocess;i++)
        {
            proc[i].at=arrivaltime[i];
            proc[i].bt=bursttime[i];
            proc[i].pr=priority[i];
            proc[i].pno=i+1;
        }

        //Using inbuilt sort function

        sort(proc,proc+totalprocess,comp);

        //Calling function findgc for finding Gantt Chart

        findgc();

        return 0;
}

// This code is contributed by Anukul Chand.
```

**Output:**

```
Process no    Start_time   Complete_time   Turn A round_Time     Waiting_Time
1                1            4               3                     0
2                5           10               8                     3
3                4            5               2                     1
4                10           17              13                    6
5                17            21             16                    12
Average Waiting Time is : 4.4
Average Turn Around time is : 8.4
```

**Time Complexity:** O(N * logN), where N is the total number of processes.
**Auxiliary Space:** O(N)