PROGRAM – 6

**AIM**: To write a program to find the average waiting and turnaround time for the following process scheduling algorithms. Input the number of processes and their details of process bursts time and arrival time from the user.
a. SJF without preemption
b. SJF with preemption
c. Priority(Added information of process priority)
d. Round Robin(Added information of quantum time)

**INTRODUCTION**: In the following program, we write a C program which can be used to implement SJF(Shortest Job First), Prioriity Scheduling and Round Robin Algorithm.
a. Shortest Job First(Without Preemption)- In this scheduling algorithm, the job with the shortest burst time is selected and it is executed till it is completed or till the time it frees CPU and makes a request for the I/O. There is no preemption involved.
b. Shortest Job First(With Preemption)- In this case, a new process that arrives and has a burst time lesser than the currently executing process can preempt the process in execution.
c. Priority Scheduling- In Priority Scheduling, each process is assigned a priority and the order in which processes get the CPU time depends on the priority assigned.
d. Round Robin Scheduling- In this scheduling algorithm, we fix a particular period of time known as Time Quantum, and every process is allowed to execute for time equal to or less than this time quantum depending on whether the burst time of the process is more than or less than the time quantum. As soon as the time quantum is elapsed, the process is added to the back of the queue and the next process is given the CPU.

**C PROGRAM :**
**a.) SJF WITHOUT PREEMPTION**

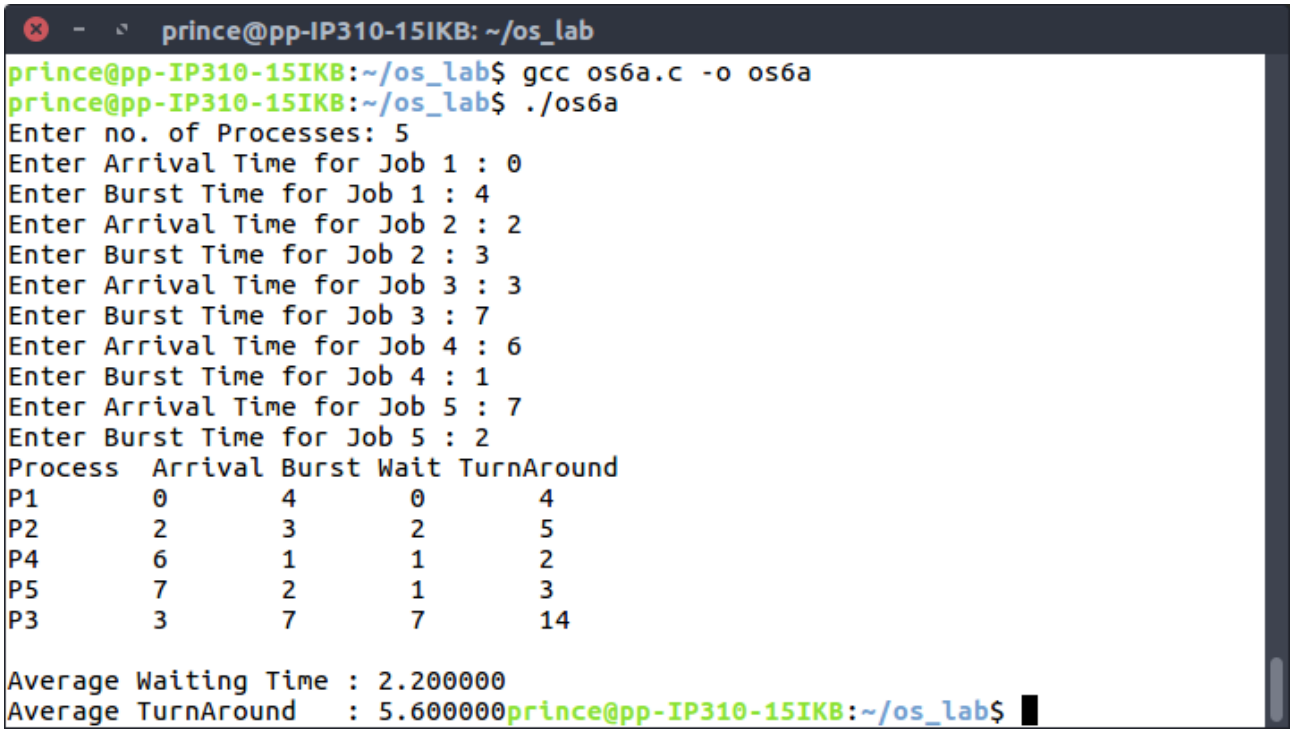
```
#include<stdio.h>
#include<stdlib.h>
```

```
void swap(int ar[],int bt[],int pid[],int wt[],int i,int j){
  int temp;

  temp=ar[i];
  ar[i]=ar[j];
  ar[j]=temp;

  temp=bt[i];
  bt[i]=bt[j];
  bt[j]=temp;

  temp=pid[i];
  pid[i]=pid[j];
  pid[j]=temp;

  temp=wt[i];
  wt[i]=wt[j];
  wt[j]=temp;
}
int partition(int ar[],int bt[],int pid[],int wt[],int p,int r){
  int x=bt[r];
  int i=p-1;
  int j;
  for(j=p;j<r;j++){
      if(bt[j]<x){
        i=i+1;
        swap(ar,bt,pid,wt,i,j);
      }
  }
  swap(ar,bt,pid,wt,i+1,r);
  return i+1;

}
void customquicksort(int ar[],int bt[],int pid[],int wt[],int p,int r) {

  if(p<r){
    int q=partition(ar,bt,pid,wt,p,r);
    customquicksort(ar,bt,pid,wt,p,q-1);
    customquicksort(ar,bt,pid,wt,q+1,r);
```

```
  }
}

int main()
{ int n;
  printf("Enter no. of Processes: ");
  scanf("%d",&n);
  int ar[n],bt[n],pid[n],wt[n],btcopy[n];
  int i,temp;
  for(i=0;i<n;i++){
    printf("Enter Arrival Time for Job %d : ",i+1 );
    scanf("%d",&ar[i]);
    printf("Enter Burst Time for Job %d : ",i+1 );
    scanf("%d",&bt[i]);
    btcopy[i]=bt[i];
    pid[i]=i+1;//p1,p2,p3...
    wt[i]=0;
  }

  int timestamp=0;
  int k,j;

  timestamp=ar[0];
  int jobCompleted=0;
  customquicksort(ar,bt,pid,wt,0,n-1);


  while(jobCompleted<n){


    for(j=0;j<n;j++){
      if(ar[j]<=timestamp&&bt[j]!=0){
        k=j;
        break;
      }
    }


    for(j=0;j<n;j++){
```

```
      if(k<n&&ar[j]<=timestamp&&bt[j]==bt[k]&&ar[j]<ar[k])
      k=j;
    }


    if(k!=n){
      wt[k]+=(timestamp-ar[k]>=0)?timestamp-ar[k]:0;
      timestamp+=bt[k];
      bt[k]=0;
      jobCompleted++;
      customquicksort(ar,bt,pid,wt,0,n-1);

    }
    else{
      timestamp+=1;
    }
      k=n;
  }


  int totalwait=0;
  int totalturnaround=0;
  printf("Process  Arrival Burst Wait TurnAround\n");
  for(k=n-1;k>=0;k--){
  printf("P%d \t %d \t %d \t %d \t %d\n",pid[k],ar[k],btcopy[pid[k]-
1],wt[k],btcopy[pid[k]-1]+wt[k]);
  totalwait+=wt[k];
  totalturnaround+=btcopy[pid[k]-1]+wt[k];
  }

  printf("\nAverage Waiting Time : %f",(double)totalwait/n);
  printf("\nAverage TurnAround   : %f",(double)totalturnaround/n);
  return 0;

}
```

**OUTPUT :**




**b.) SJF WITH PREEMPTION**

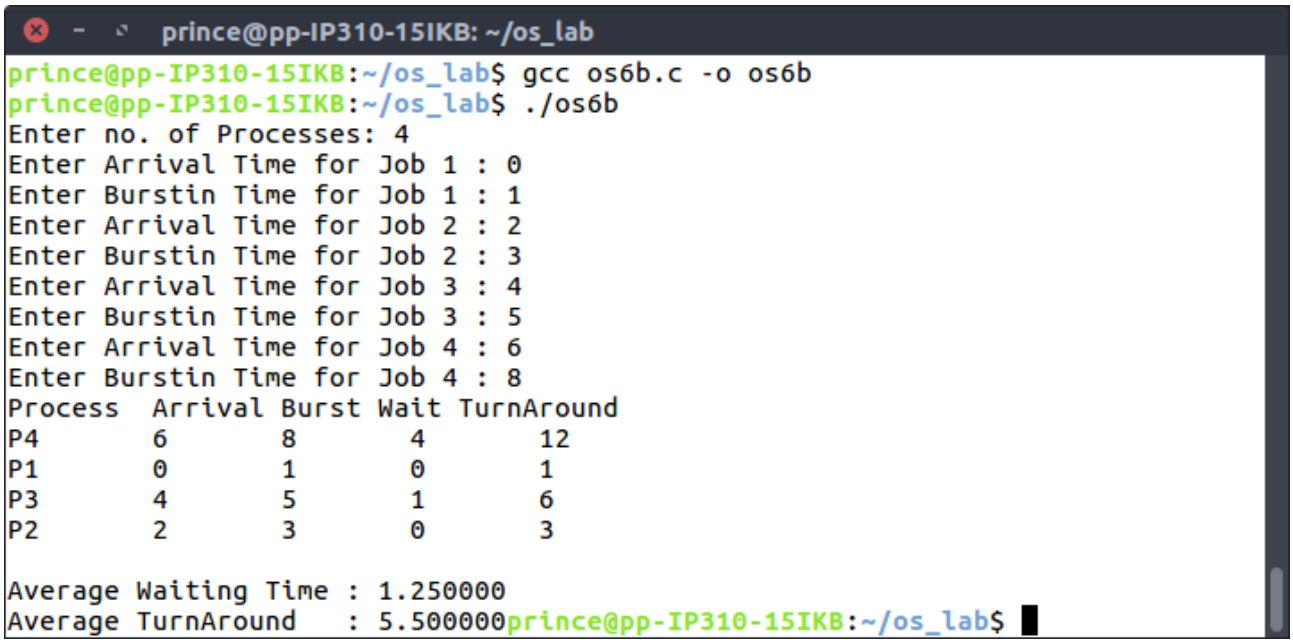
```
#include<stdio.h>
#include<stdlib.h>
void swap(int ar[],int bt[],int pid[],int wt[],int i,int j){
  int temp;

  temp=ar[i];
  ar[i]=ar[j];
  ar[j]=temp;

  temp=bt[i];
  bt[i]=bt[j];
  bt[j]=temp;

  temp=pid[i];
  pid[i]=pid[j];
  pid[j]=temp;

  temp=wt[i];
  wt[i]=wt[j];
  wt[j]=temp;
}
```

```
int partition(int ar[],int bt[],int pid[],int wt[],int p,int r){
  int x=bt[r];
  int i=p-1;
  int j;
  for(j=p;j<r;j++){
      if(bt[j]<x){
        i=i+1;
        swap(ar,bt,pid,wt,i,j);
      }
  }
  swap(ar,bt,pid,wt,i+1,r);
  return i+1;
}

void customquicksort(int ar[],int bt[],int pid[],int wt[],int p,int r) {
  if(p<r){
    int q=partition(ar,bt,pid,wt,p,r);
    customquicksort(ar,bt,pid,wt,p,q-1);
    customquicksort(ar,bt,pid,wt,q+1,r);
  }
}

int main()
{ int n;
  printf("Enter no. of Processes: ");
  scanf("%d",&n);
  int ar[n],bt[n],pid[n],wt[n],btcopy[n];
  int i;
  for(i=0;i<n;i++){
    printf("Enter Arrival Time for Job %d : ",i+1 );
    scanf("%d",&ar[i]);
    printf("Enter Burstin Time for Job %d : ",i+1 );
    scanf("%d",&bt[i]);
    btcopy[i]=bt[i];
    pid[i]=i+1;
    wt[i]=0;
  }
  int timestamp=ar[0];
```

```
int jobCompleted=0;
int j=0,k=n;
i=0;

customquicksort(ar,bt,pid,wt,0,n-1);

while(jobCompleted<n){


 for(j=0;j<n;j++){
  if(ar[j]<=timestamp&&bt[j]!=0){
    k=j;
    break;
  }
 }


 for(j=0;j<n;j++){
  if(k<n&&ar[j]<=timestamp&&bt[j]==bt[k]&&ar[j]<ar[k])
  k=j;
 }


 for(j=0;j<n;j++){
  if(k<n&&ar[j]<=timestamp&&bt[j]!=0&&j!=k)
  wt[j]++;
 }


 timestamp++;


 if(k!=n){
  bt[k]--;
  if(bt[k]==0)
  jobCompleted++;
  customquicksort(ar,bt,pid,wt,0,n-1);
 }
```

```
  k=n;
 }

 int totalwait=0;
 int totalturnaround=0;
 printf("Process  Arrival Burst Wait TurnAround\n");
 for(k=0;k<n;k++){
 printf("P%d \t %d \t %d \t %d \t %d\n",pid[k],ar[k],btcopy[pid[k]-
1],wt[k],btcopy[pid[k]-1]+wt[k]);
 totalwait+=wt[k];
 totalturnaround+=btcopy[pid[k]-1]+wt[k];
 }

 printf("\nAverage Waiting Time : %f",(double)totalwait/n);
 printf("\nAverage TurnAround   : %f",(double)totalturnaround/n);

 return 0;

}
```

**OUTPUT:**




**c.) PRIORITY**

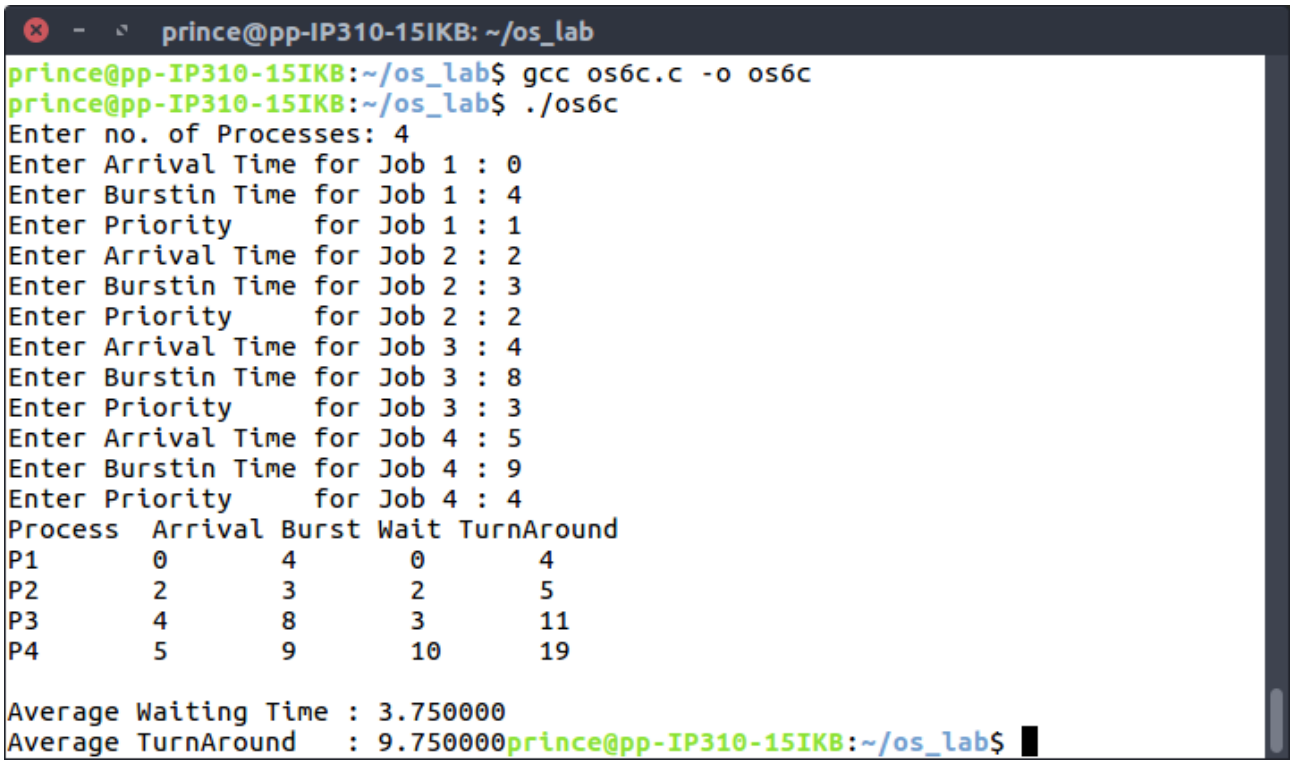
```
#include<stdio.h>
#include<sys/wait.h>
```

```
void swap(int ar[],int bt[],int pid[],int wt[],int i,int j){
  int temp;

  temp=ar[i];
  ar[i]=ar[j];
  ar[j]=temp;

  temp=bt[i];
  bt[i]=bt[j];
  bt[j]=temp;

  temp=pid[i];
  pid[i]=pid[j];
  pid[j]=temp;

  temp=wt[i];
  wt[i]=wt[j];
  wt[j]=temp;
}

int partition(int ar[],int bt[],int pid[],int wt[],int p,int r){
  int x=bt[r];
  int i=p-1;
  int j;
  for(j=p;j<r;j++){
      if(bt[j]<x){
        i=i+1;
        swap(ar,bt,pid,wt,i,j);
      }
  }
  swap(ar,bt,pid,wt,i+1,r);
  return i+1;

}
void customquicksort(int ar[],int bt[],int pid[],int wt[],int p,int r) {

  if(p<r){
    int q=partition(ar,bt,pid,wt,p,r);
    customquicksort(ar,bt,pid,wt,p,q-1);
```

```
    customquicksort(ar,bt,pid,wt,q+1,r);
  }
}

int main(){
  int n;
  printf("Enter no. of Processes: ");
  scanf("%d",&n);
  int ar[n],bt[n],pid[n],wt[n],btcopy[n],priority[n];
  int i;
  for(i=0;i<n;i++){
    printf("Enter Arrival Time for Job %d : ",i+1 );
    scanf("%d",&ar[i]);
    printf("Enter Burstin Time for Job %d : ",i+1 );
    scanf("%d",&bt[i]);
    printf("Enter Priority     for Job %d : ",i+1 );
    scanf("%d",&priority[i]);
    btcopy[i]=bt[i];
    pid[i]=i+1;
    wt[i]=0;
  }
  int timestamp=ar[0];
  int jobCompleted=0;
  int j=0,k=n;
  i=0;


  customquicksort(ar,priority,pid,bt,0,n-1);

  while(jobCompleted<n){


    for(j=0;j<n;j++){
      if(ar[j]<=timestamp&&bt[j]!=0){
        k=j;
        break;
      }
    }
```

```
    for(j=0;j<n;j++){
      if(k<n&&ar[j]<=timestamp&&priority[j]==priority[k]&&ar[j]<ar[k])
      k=j;
    }


    for(j=0;j<n;j++){
      if(k<n&&ar[j]<=timestamp&&bt[j]!=0&&j!=k)
      wt[j]++;
    }


    timestamp++;
    if(k!=n){
      bt[k]--;
      if(bt[k]==0)
      jobCompleted++;
    }

    k=n;
  }

  int totalwait=0;
  int totalturnaround=0;
  printf("Process  Arrival Burst Wait TurnAround\n");
  for(k=0;k<n;k++){
    printf("P%d \t %d \t %d \t %d \t %d\n",pid[k],ar[k],btcopy[pid[k]-
1],wt[k],btcopy[pid[k]-1]+wt[k]);
    totalwait+=wt[k];
    totalturnaround+=btcopy[pid[k]-1]+wt[k];
  }

  printf("\nAverage Waiting Time : %f",(double)totalwait/n);
  printf("\nAverage TurnAround   : %f",(double)totalturnaround/n);
  return 0;

}
```

**OUTPUT:**




**d.) ROUND ROBIN**

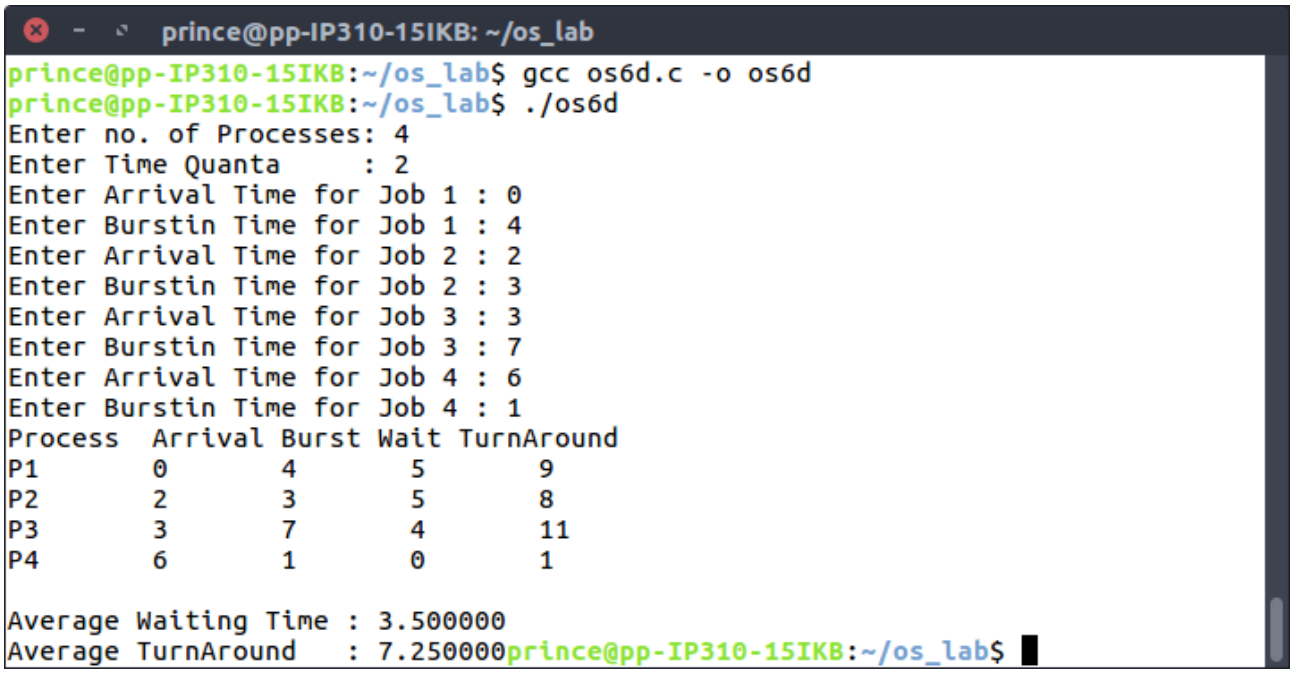
```
#include<stdio.h>
#include<sys/wait.h>
void swap(int ar[],int bt[],int pid[],int wt[],int i,int j){
  int temp;

  temp=ar[i];
  ar[i]=ar[j];
  ar[j]=temp;

  temp=bt[i];
  bt[i]=bt[j];
  bt[j]=temp;

  temp=pid[i];
  pid[i]=pid[j];
  pid[j]=temp;

  temp=wt[i];
  wt[i]=wt[j];
  wt[j]=temp;
}
```

```
int partition(int ar[],int bt[],int pid[],int wt[],int p,int r){
  int x=bt[r];
  int i=p-1;
  int j;
  for(j=p;j<r;j++){
      if(bt[j]<x){
        i=i+1;
        swap(ar,bt,pid,wt,i,j);
      }
  }
  swap(ar,bt,pid,wt,i+1,r);
  return i+1;

}
void customquicksort(int ar[],int bt[],int pid[],int wt[],int p,int r) {

  if(p<r){
    int q=partition(ar,bt,pid,wt,p,r);
    customquicksort(ar,bt,pid,wt,p,q-1);
    customquicksort(ar,bt,pid,wt,q+1,r);
  }
}
int main(){
  int n;
  printf("Enter no. of Processes: ");
  scanf("%d",&n);
  int ar[n],bt[n],pid[n],wt[n],btcopy[n],tq;
  printf("Enter Time Quanta     : ");
  scanf("%d",&tq);
  int i;
  for(i=0;i<n;i++){
    printf("Enter Arrival Time for Job %d : ",i+1 );
    scanf("%d",&ar[i]);
    printf("Enter Burstin Time for Job %d : ",i+1 );
    scanf("%d",&bt[i]);
    btcopy[i]=bt[i];
    pid[i]=i+1;
    wt[i]=0;
  }
```

```
int timestamp=ar[0];
int jobCompleted=0;
int j=0,k=n;
i=0;

while(jobCompleted<n){

  for(j=0;j<n;j++){

    if(bt[j]>0&&ar[j]<=timestamp){

      for(i=0;i<n;i++){
        if(i!=j&&ar[i]<=timestamp&&bt[i]!=0)
        wt[i]+=(bt[j]-tq)>0?tq:(bt[j]-tq==0)?tq:bt[j];
      }
      bt[j]=(bt[j]-tq);
      if(bt[j]==0){
        timestamp+=tq;
        jobCompleted++;
      }
      else if(bt[j]<0){
        bt[j]+=tq;
        timestamp+=bt[j];
        bt[j]=0;
        jobCompleted++;
      }
      else{
        timestamp+=tq;
      }
    }
    else if(ar[j]>timestamp){
      if(jobCompleted==(j))
      timestamp++;
      break;
    }

  }
}
```

```
  int totalwait=0;
  int totalturnaround=0;
  printf("Process  Arrival Burst Wait TurnAround\n");
  for(k=0;k<n;k++){
    printf("P%d \t %d \t %d \t %d \t %d\n",pid[k],ar[k],btcopy[pid[k]-
1],wt[k],btcopy[pid[k]-1]+wt[k]);
    totalwait+=wt[k];
    totalturnaround+=btcopy[pid[k]-1]+wt[k];
  }

  printf("\nAverage Waiting Time : %f",(double)totalwait/n);
  printf("\nAverage TurnAround   : %f",(double)totalturnaround/n);
return 0;

}
```

**OUTPUT:**




**LEARNING OUTCOMES**

In the above program, we came to know about the various scheduling algorithms and the mechanism followed to schedule the processes according to the algos. We calculated the average waiting time and turnaround time in the above cases which are measures of performance of the algorithms.