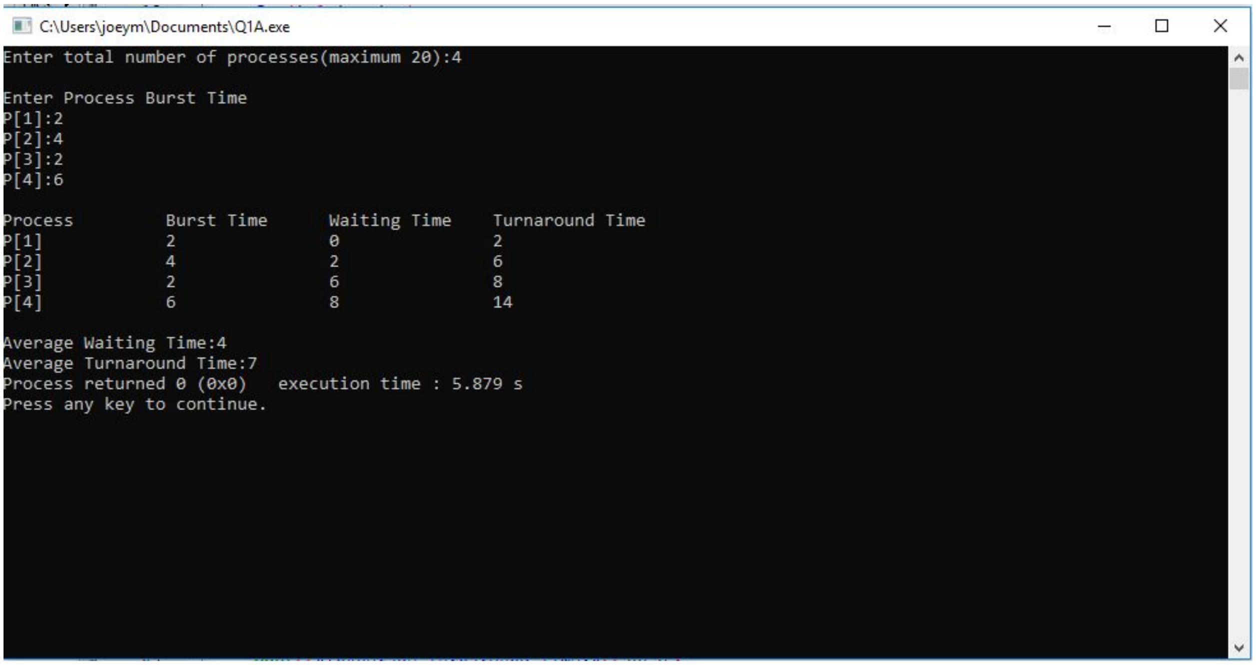
Q1. (FCFS)

```
#include<stdio.h>

int main()
{
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("Enter total number of processes(maximum 20):");
   scanf("%d",&n);

   printf("\nEnter Process Burst Time\n");
   for(i=0;i<n;i++)
   {
       printf("P[%d]:",i+1);
       scanf("%d",&bt[i]);
   }

   wt[0]=0;    //waiting time for first process is 0

   //calculating waiting time
   for(i=1;i<n;i++)
   {
       wt[i]=0;
       for(j=0;j<i;j++)
           wt[i]+=bt[j];
   }

   printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

   //calculating turnaround time
   for(i=0;i<n;i++)
   {
       tat[i]=bt[i]+wt[i];
       avwt+=wt[i];
       avtat+=tat[i];
       printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
   }
```

```
    avwt/=i;
    avtat/=i;
    printf("\n\nAverage Waiting Time:%d",avwt);
    printf("\nAverage Turnaround Time:%d",avtat);

    return 0;
}
```

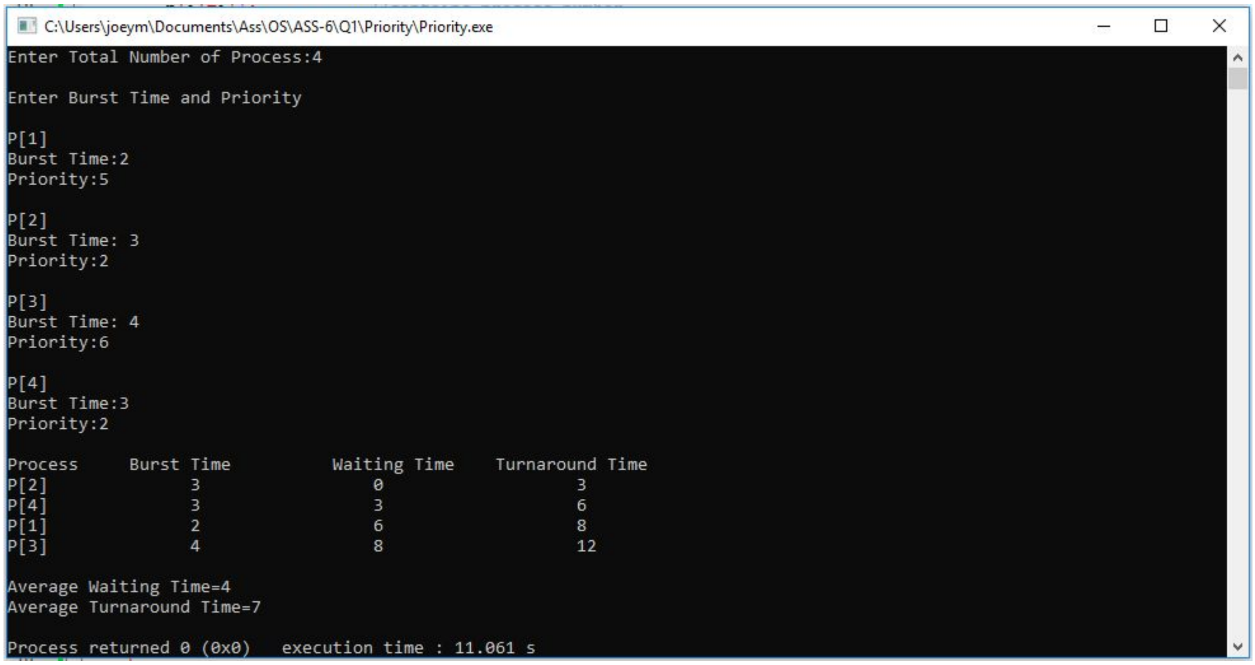


```
Enter total number of processes(maximum 20):4

Enter Process Burst Time
P[1]:2
P[2]:4
P[3]:2
P[4]:6

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            2               0               2
P[2]            4               2               6
P[3]            2               6               8
P[4]            6               8               14

Average Waiting Time:4
Average Turnaround Time:7
Process returned 0 (0x0)   execution time : 5.879 s
Press any key to continue.
```

(Priority)

```
#include<stdio.h>

int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1;           //contains process number
    }

    //sorting burst time, priority and process number in ascending order
using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(pr[j]<pr[pos])
                pos=j;
        }

        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;    //waiting time for first process is zero

    //calculate waiting time
    for(i=1;i<n;i++)
    {
```

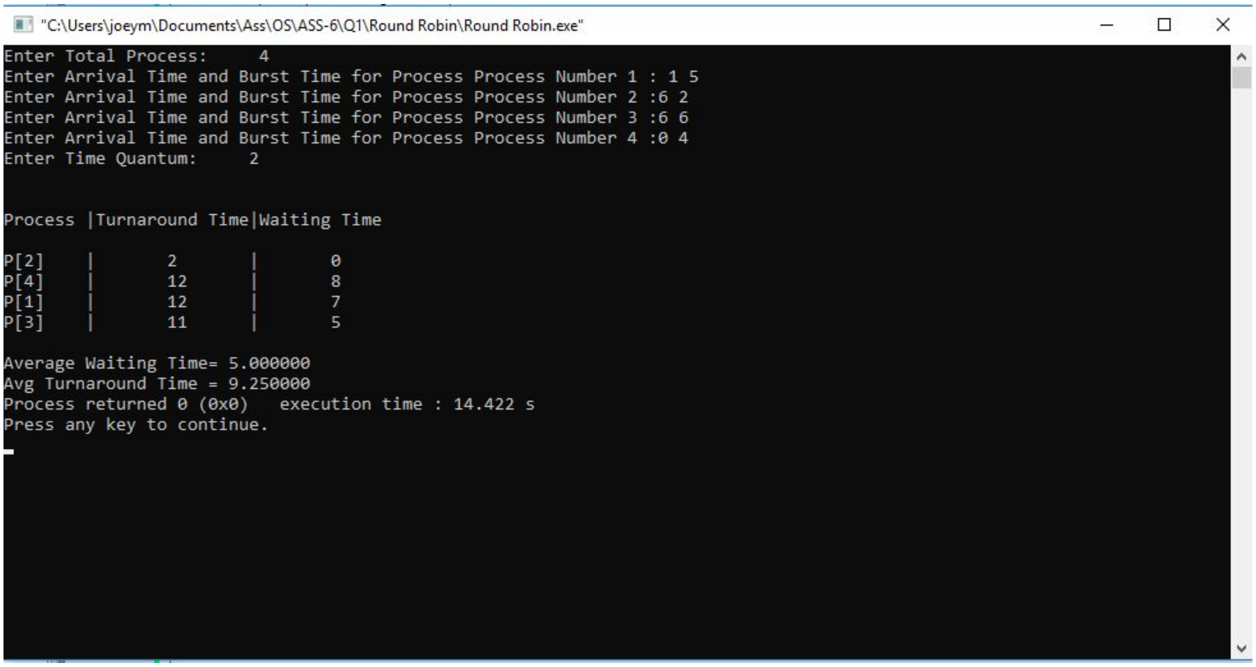
```c
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=total/n;      //average waiting time
    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];     //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t  %d\t\t
%d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);

    return 0;
}
```


C:\Users\joeym\Documents\Ass\OS\ASS-6\Q1\Priority\Priority.exe

```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:2
Priority:5

P[2]
Burst Time: 3
Priority:2

P[3]
Burst Time: 4
Priority:6

P[4]
Burst Time:3
Priority:2

Process     Burst Time          Waiting Time    Turnaround Time
P[2]              3                 0                   3
P[4]              3                 3                   6
P[1]              2                 6                   8
P[3]              4                 8                   12

Average Waiting Time=4
Average Turnaround Time=7

Process returned 0 (0x0)   execution time : 11.061 s
```

(Round Robin)

```
#include<stdio.h>

int main()
{

 int count,j,n,time,remain,flag=0,time_quantum, min=0, starter=0;
 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 and 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\nProcess\t|Turnaround Time|Waiting Time\n\n");
 min = at[0];
 for(int i = 1; i < n; i++) {
   if (at[i] <= min) {
       min = at[i];
       starter = i;  }
 }
 for(time=min,count=starter;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_quantum;
   }
   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]-bt[count]);
     wait_time+=time-at[count]-bt[count];
     turnaround_time+=time-at[count];
     flag=0;
   }
   if(count==n-1) {
     count=0; }
   else if(at[count+1]<=time) {
     count++; }
   else {
     count=0; }
 }
 printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
 printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

 return 0;
}
```



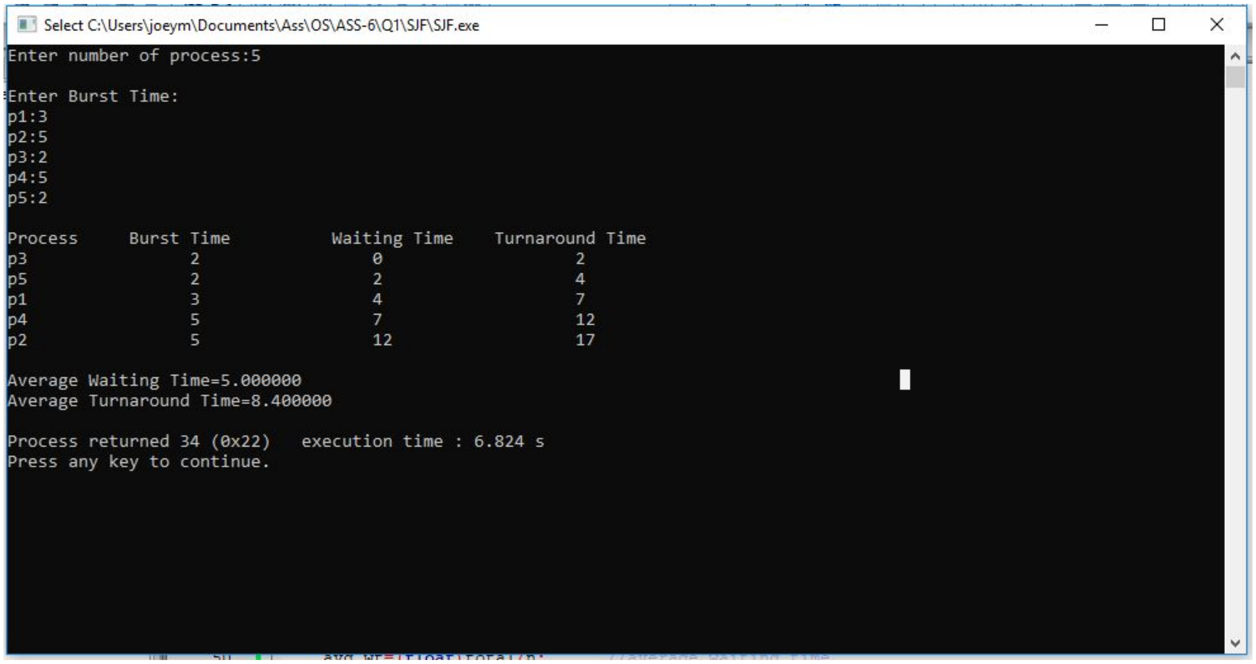
"C:\Users\joeym\Documents\Ass\OS\ASS-6\Q1\Round Robin\Round Robin.exe"

```
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1 : 1 5
Enter Arrival Time and Burst Time for Process Process Number 2 :6 2
Enter Arrival Time and Burst Time for Process Process Number 3 :6 6
Enter Arrival Time and Burst Time for Process Process Number 4 :0 4
Enter Time Quantum:     2


Process |Turnaround Time|Waiting Time

P[2]    |       2       |       0
P[4]    |       12      |       8
P[1]    |       12      |       7
P[3]    |       11      |       5

Average Waiting Time= 5.000000
Avg Turnaround Time = 9.250000
Process returned 0 (0x0)   execution time : 14.422 s
Press any key to continue.
```

(SJF)

```
#include<stdio.h>

void main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;           //contains process number
    }

    //sorting burst time in ascending order using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }

    wt[0]=0;            //waiting time for first process will be zero

    //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }

    avg_wt=(float)total/n;      //average waiting time
    total=0;
```

```
    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];     //calculate turnaround time
        total+=tat[i];
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=(float)total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

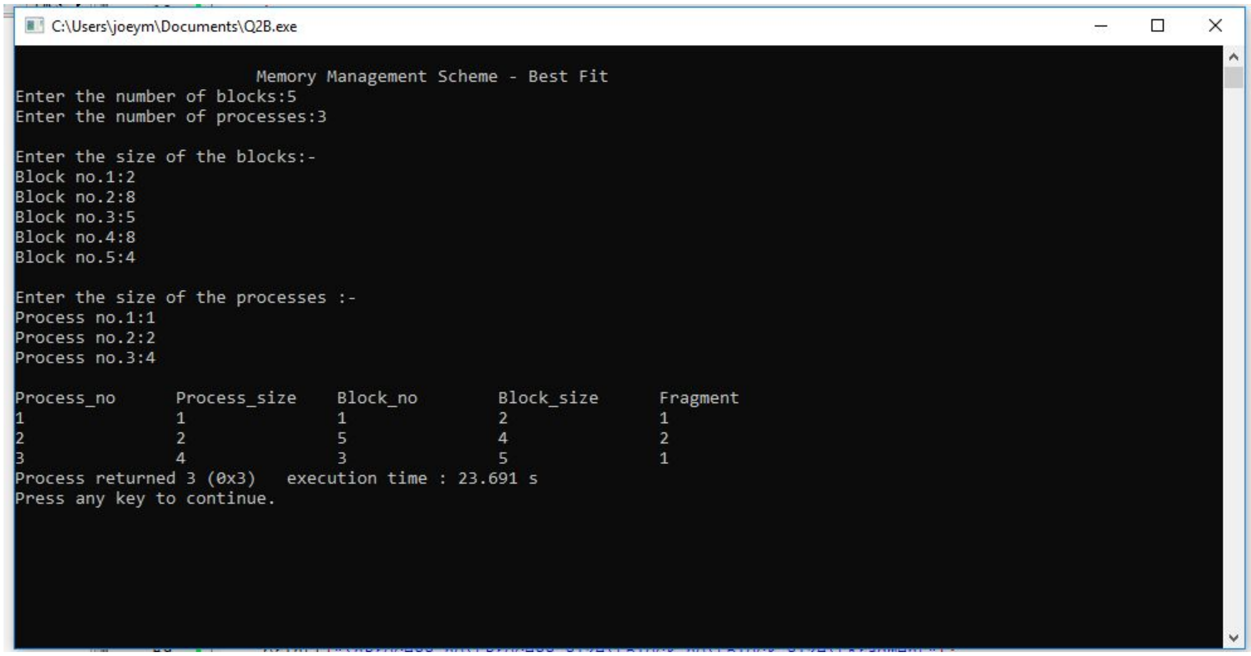


```
Enter number of process:5

Enter Burst Time:
p1:3
p2:5
p3:2
p4:5
p5:2

Process     Burst Time          Waiting Time    Turnaround Time
p3                2                 0                   2
p5                2                 2                   4
p1                3                 4                   7
p4                5                 7                   12
p2                5                 12                  17

Average Waiting Time=5.000000
Average Turnaround Time=8.400000

Process returned 34 (0x22)   execution time : 6.824 s
Press any key to continue.
```

Q2.) (BEST FIT)

```c
#include<stdio.h>

void main()
{
   int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
   static int barray[20],parray[20];

   printf("\n\t\t\tMemory Management Scheme - Best Fit");
   printf("\nEnter the number of blocks:");
   scanf("%d",&nb);
   printf("Enter the number of processes:");
   scanf("%d",&np);

   printf("\nEnter the size of the blocks:-\n");
   for(i=1;i<=nb;i++)
   {
       printf("Block no.%d:",i);
       scanf("%d",&b[i]);
   }

   printf("\nEnter the size of the processes :-\n");
   for(i=1;i<=np;i++)
   {
       printf("Process no.%d:",i);
       scanf("%d",&p[i]);
   }

   for(i=1;i<=np;i++)
   {
       for(j=1;j<=nb;j++)
       {
           if(barray[j]!=1)
           {
               temp=b[j]-p[i];
               if(temp>=0)
                   if(lowest>temp)
```

```
                    {
                         parray[i]=j;
                         lowest=temp;
                    }
               }
          }

          fragment[i]=lowest;
          barray[parray[i]]=1;
          lowest=10000;
     }

     printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
     for(i=1;i<=np && parray[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```



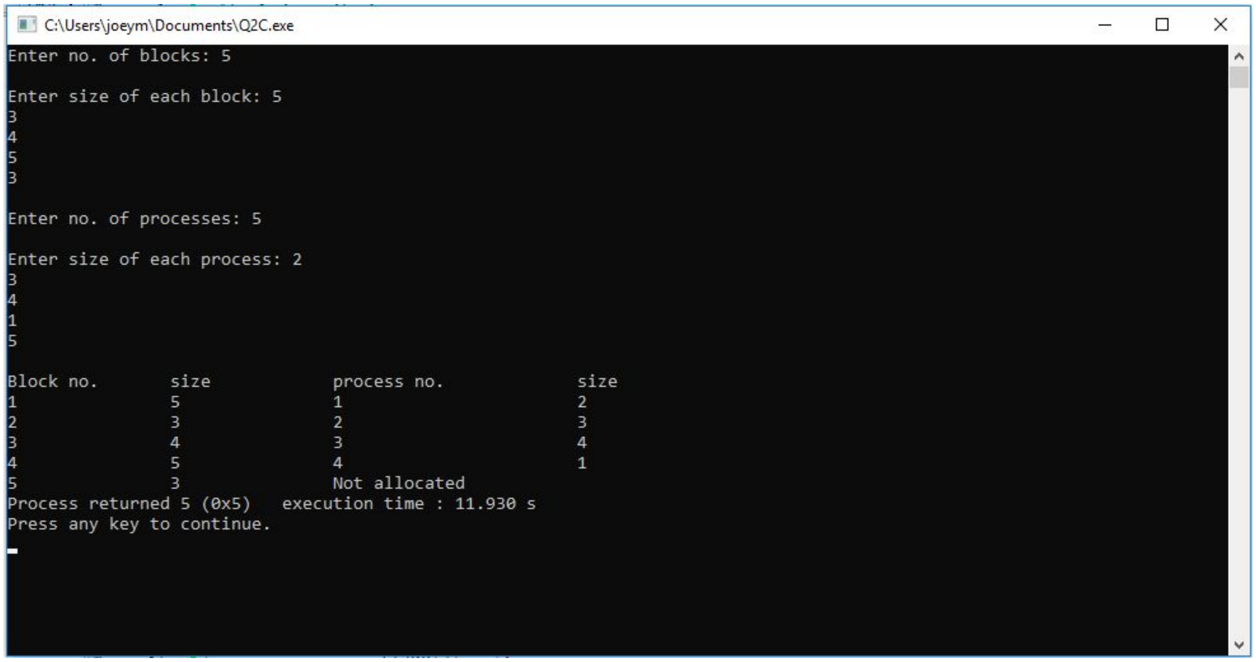
```
                        Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:3

Enter the size of the blocks:-
Block no.1:2
Block no.2:8
Block no.3:5
Block no.4:8
Block no.5:4

Enter the size of the processes :-
Process no.1:1
Process no.2:2
Process no.3:4

Process_no      Process_size    Block_no        Block_size      Fragment
1               1               1               2               1
2               2               5               4               2
3               4               3               5               1
Process returned 3 (0x3)   execution time : 23.691 s
Press any key to continue.
```

(FIRST FIT)

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

```c
void main()
{
   int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

   for(i = 0; i < 10; i++)
   {
       flags[i] = 0;
       allocation[i] = -1;
   }

   printf("Enter no. of blocks: ");
   scanf("%d", &bno);

   printf("\nEnter size of each block: ");
   for(i = 0; i < bno; i++)
       scanf("%d", &bsize[i]);

   printf("\nEnter no. of processes: ");
   scanf("%d", &pno);

   printf("\nEnter size of each process: ");
   for(i = 0; i < pno; i++)
       scanf("%d", &psize[i]);
   for(i = 0; i < pno; i++)          //allocation as per first fit
       for(j = 0; j < bno; j++)
           if(flags[j] == 0 && bsize[j] >= psize[i])
           {
               allocation[j] = i;
               flags[j] = 1;
               break;
           }

   //display allocation details
   printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
   for(i = 0; i < bno; i++)
   {
       printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
```

```
        if(flags[i] == 1)
            printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
        else
            printf("Not allocated");
    }
}
```

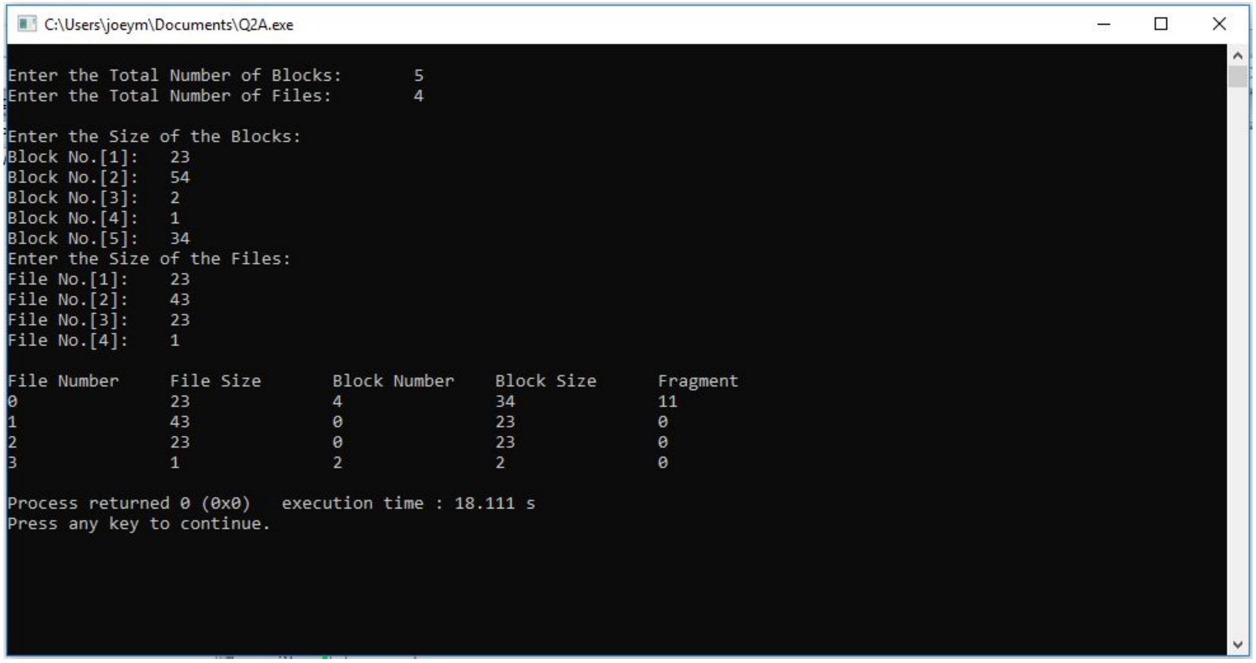


```
Enter no. of blocks: 5

Enter size of each block: 5
3
4
5
3

Enter no. of processes: 5

Enter size of each process: 2
3
4
1
5

Block no.       size            process no.             size
1               5               1                       2
2               3               2                       3
3               4               3                       4
4               5               4                       1
5               3               Not allocated
Process returned 5 (0x5)   execution time : 11.930 s
Press any key to continue.
```

(WORST FIT)

```
#include<stdio.h>

int main()
{
    int fragments[10], blocks[10], files[10];
    int m, n, number_of_blocks, number_of_files, temp, top = 0;
    static int block_arr[10], file_arr[10];
    printf("\nEnter the Total Number of Blocks:\t");
    scanf("%d",&number_of_blocks);
    printf("Enter the Total Number of Files:\t");
    scanf("%d",&number_of_files);
    printf("\nEnter the Size of the Blocks:\n");
    for(m = 0; m < number_of_blocks; m++)
    {
        printf("Block No.[%d]:\t", m + 1);
        scanf("%d", &blocks[m]);
    }
    printf("Enter the Size of the Files:\n");
    for(m = 0; m < number_of_files; m++)
    {
        printf("File No.[%d]:\t", m + 1);
        scanf("%d", &files[m]);
    }
    for(m = 0; m < number_of_files; m++)
    {
        for(n = 0; n < number_of_blocks; n++)
        {
            if(block_arr[n] != 1)
            {
                temp = blocks[n] - files[m];
                if(temp >= 0)
                {
                    if(top < temp)
                    {
                        file_arr[m] = n;
                        top = temp;
                    }
                }
            }
            fragments[m] = top;
            block_arr[file_arr[m]] = 1;
            top = 0;
        }
    }
    printf("\nFile Number\tFile Size\tBlock Number\tBlock 
Size\tFragment");
    for(m = 0; m < number_of_files; m++)
    {
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", m, files[m], 
file_arr[m], blocks[file_arr[m]], fragments[m]);
```

```
        }
        printf("\n");
        return 0;
}
```



```

Enter the Total Number of Blocks:       5
Enter the Total Number of Files:        4

Enter the Size of the Blocks:
Block No.[1]:   23
Block No.[2]:   54
Block No.[3]:   2
Block No.[4]:   1
Block No.[5]:   34
Enter the Size of the Files:
File No.[1]:    23
File No.[2]:    43
File No.[3]:    23
File No.[4]:    1

File Number     File Size       Block Number    Block Size      Fragment
0               23              4               34              11
1               43              0               23              0
2               23              0               23              0
3               1               2               2               0

Process returned 0 (0x0)   execution time : 18.111 s
Press any key to continue.
```

Q3.)

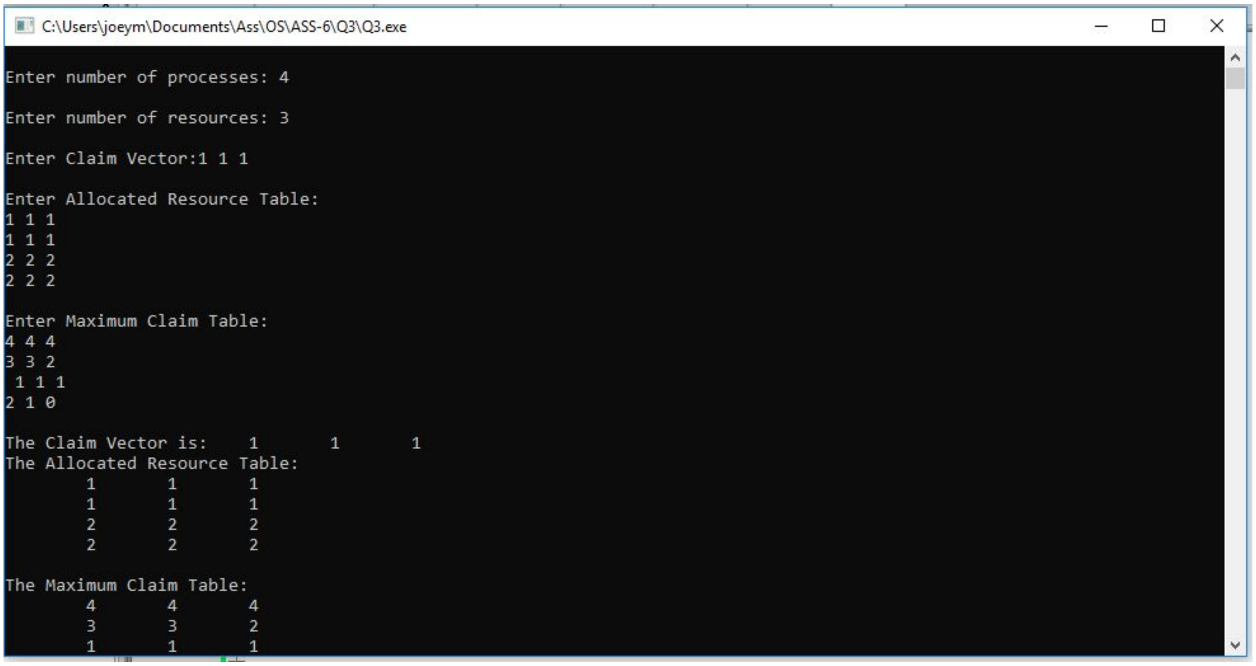
```
#include <stdio.h>

int current[5][5], maximum_claim[5][5], available[5];
int allocation[5] = {0, 0, 0, 0, 0};
int maxres[5], running[5], safe = 0;
int counter = 0, i, j, exec, resources, processes, k = 1;

int main()
{
   printf("\nEnter number of processes: ");
       scanf("%d", &processes);

       for (i = 0; i < processes; i++)
   {
           running[i] = 1;
           counter++;
       }

       printf("\nEnter number of resources: ");
       scanf("%d", &resources);

       printf("\nEnter Claim Vector:");
       for (i = 0; i < resources; i++)
   {
           scanf("%d", &maxres[i]);
       }

      printf("\nEnter Allocated Resource Table:\n");
       for (i = 0; i < processes; i++)
   {
           for(j = 0; j < resources; j++)
       {
             scanf("%d", &current[i][j]);
           }
       }
```

```
    printf("\nEnter Maximum Claim Table:\n");
    for (i = 0; i < processes; i++)
{
        for(j = 0; j < resources; j++)
    {
                scanf("%d", &maximum_claim[i][j]);
        }
    }

printf("\nThe Claim Vector is: ");
    for (i = 0; i < resources; i++)
{
        printf("\t%d", maxres[i]);
}

    printf("\nThe Allocated Resource Table:\n");
    for (i = 0; i < processes; i++)
{
        for (j = 0; j < resources; j++)
    {
                printf("\t%d", current[i][j]);
        }
    printf("\n");
    }

    printf("\nThe Maximum Claim Table:\n");
    for (i = 0; i < processes; i++)
{
        for (j = 0; j < resources; j++)
    {
            printf("\t%d", maximum_claim[i][j]);
        }
        printf("\n");
    }

    for (i = 0; i < processes; i++)
{
```

```
           for (j = 0; j < resources; j++)
       {
                   allocation[j] += current[i][j];
           }
       }

       printf("\nAllocated resources:");
       for (i = 0; i < resources; i++)
   {
           printf("\t%d", allocation[i]);
       }

       for (i = 0; i < resources; i++)
   {
           available[i] = maxres[i] - allocation[i];
   }

       printf("\nAvailable resources:");
       for (i = 0; i < resources; i++)
   {
           printf("\t%d", available[i]);
       }
       printf("\n");

       while (counter != 0)
   {
           safe = 0;
           for (i = 0; i < processes; i++)
       {
                   if (running[i])
           {
                       exec = 1;
                       for (j = 0; j < resources; j++)
               {
                               if (maximum_claim[i][j] - current[i][j] >
available[j])
                   {
```

```
                  exec = 0;
                  break;
                }
            }
            if (exec)
        {
                printf("\nProcess%d is executing\n", i +
1);

                running[i] = 0;
                counter--;
                safe = 1;

                for (j = 0; j < resources; j++)
          {
                  available[j] += current[i][j];
                }
              break;
            }
          }
      }
      if (!safe)
    {
          printf("\nThe processes are in unsafe state.\n");
          break;
      }
    else
    {
          printf("\nThe process is in safe state");
          printf("\nAvailable vector:");

          for (i = 0; i < resources; i++)
      {
            printf("\t%d", available[i]);
          }

        printf("\n");
      }
```

```
          }
          return 0;
}
```

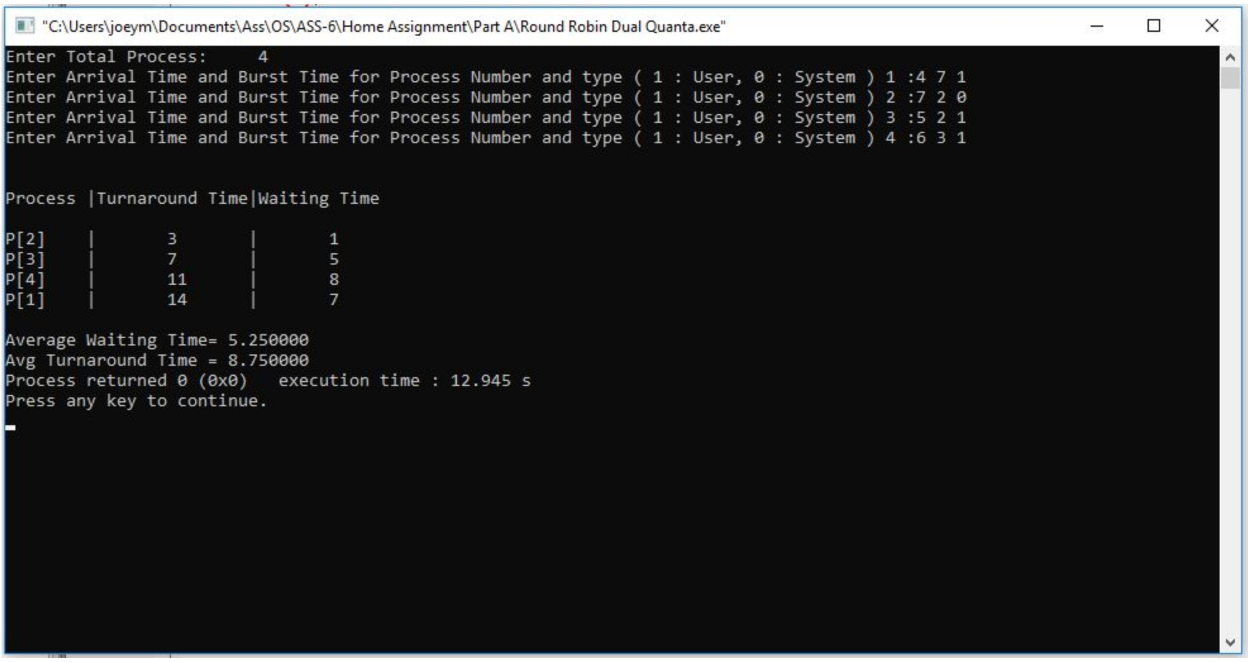


```
Enter number of processes: 4

Enter number of resources: 3

Enter Claim Vector:1 1 1

Enter Allocated Resource Table:
1 1 1
1 1 1
2 2 2
2 2 2

Enter Maximum Claim Table:
4 4 4
3 3 2
 1 1 1
2 1 0

The Claim Vector is:    1       1       1
The Allocated Resource Table:
        1       1       1
        1       1       1
        2       2       2
        2       2       2

The Maximum Claim Table:
        4       4       4
        3       3       2
        1       1       1
```

Q4.)

A.)

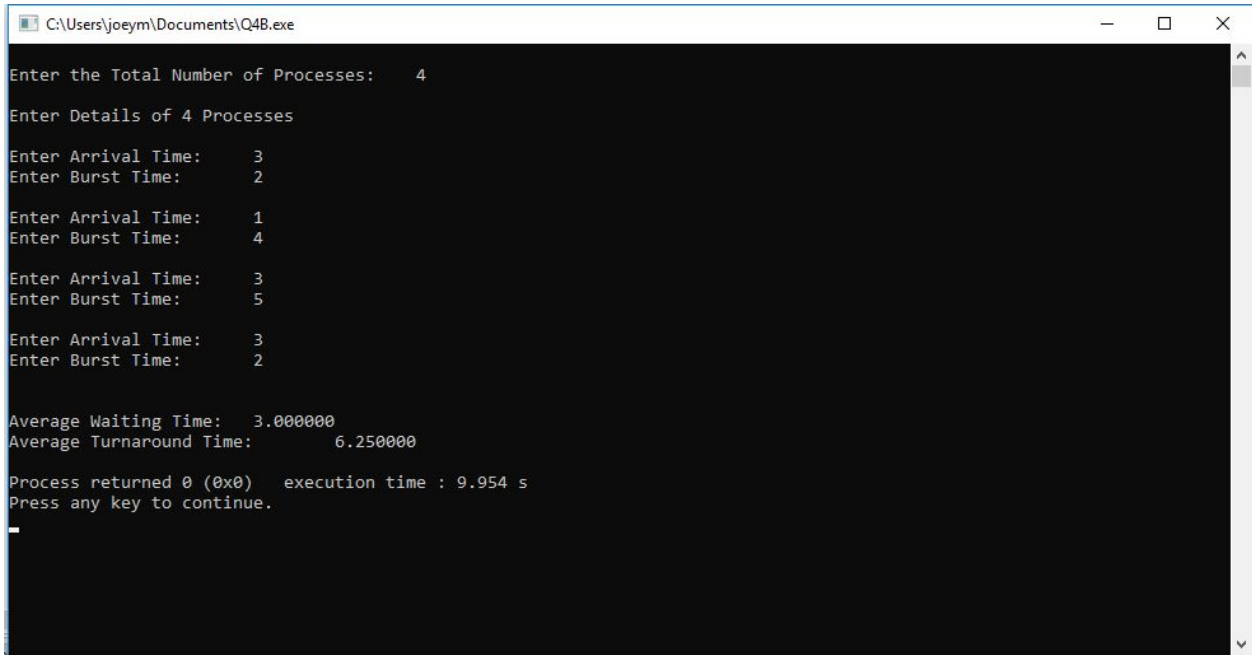
```
#include<stdio.h>

int main()
{

 int count,j,n,time,remain,flag=0, time_quanta_user = 2,
time_quanta_system = 5, min=0, starter=0;
 int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10], type[10];
 printf("Enter Total Process:\t ");
 scanf("%d",&n);
 remain=n;
 for(count=0;count<n;count++)
 {
   printf("Enter Arrival Time and Burst Time for Process Number and type (
1 : User, 0 : System ) %d :",count+1);
   scanf("%d",&at[count]);
   scanf("%d",&bt[count]);
   scanf("%d", &type[count]);
   rt[count]=bt[count];
 }
 printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
 min = at[0];
 for(int i = 1; i < n; i++) {
   if (at[i] <= min) {
       min = at[i];
       starter = i;  }
 }
 for(time=min,count=starter;remain!=0;)
 {
   if(((type[count] == 1) && (rt[count]<=time_quanta_user)) &&
rt[count]>0)
   {
     time+=rt[count];
     rt[count]=0;
     flag=1;
   }
   else if(((type[count] == 0) && (rt[count]<=time_quanta_system)) &&
rt[count]>0)
   {
     time+=rt[count];
     rt[count]=0;
     flag=1;
   }
   else if(rt[count]>0)
   {
     if(type[count]==1) {
       rt[count]-=time_quanta_user;
       time+=time_quanta_user;
     }
     else {
       rt[count]-=time_quanta_system;
       time+=time_quanta_system;
     }
   }
   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]-bt[co
unt]);
     wait_time+=(time-at[count])-bt[count];
     turnaround_time+=time-at[count];
     flag=0;
   }
   if(count==n-1) {
     count=0; }
   else if(at[count+1]<=time) {
     count++; }
   else {
     count=0; }
 }
 printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
 printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

 return 0;
}
```


"C:\Users\joeym\Documents\Ass\OS\ASS-6\Home Assignment\Part A\Round Robin Dual Quanta.exe"

```
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Number and type ( 1 : User, 0 : System ) 1 :4 7 1
Enter Arrival Time and Burst Time for Process Number and type ( 1 : User, 0 : System ) 2 :7 2 0
Enter Arrival Time and Burst Time for Process Number and type ( 1 : User, 0 : System ) 3 :5 2 1
Enter Arrival Time and Burst Time for Process Number and type ( 1 : User, 0 : System ) 4 :6 3 1


Process |Turnaround Time|Waiting Time

P[2]    |       3       |       1
P[3]    |       7       |       5
P[4]    |       11      |       8
P[1]    |       14      |       7

Average Waiting Time= 5.250000
Avg Turnaround Time = 8.750000
Process returned 0 (0x0)   execution time : 12.945 s
Press any key to continue.
```

B.)

```
#include <stdio.h>

int main()
{
      int arrival_time[10], burst_time[10], temp[10];
      int i, smallest, count = 0, time, limit;
      double wait_time = 0, turnaround_time = 0, end;
      float average_waiting_time, average_turnaround_time;
      printf("\nEnter the Total Number of Processes:\t");
      scanf("%d", &limit);
      printf("\nEnter Details of %d Processes\n", limit);
      for(i = 0; i < limit; i++)
      {
        printf("\nEnter Arrival Time:\t");
        scanf("%d", &arrival_time[i]);
        printf("Enter Burst Time:\t");
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    burst_time[9] = 9999;
    for(time = 0; count != limit; time++)
    {
        smallest = 9;
        for(i = 0; i < limit; i++)
        {
            if(arrival_time[i] <= time && burst_time[i] <
burst_time[smallest] && burst_time[i] > 0)
            {
                smallest = i;
            }
        }
        burst_time[smallest]--;
        if(burst_time[smallest] == 0)
        {
            count++;
            end = time + 1;
            wait_time = wait_time + end - arrival_time[smallest] -
temp[smallest];
            turnaround_time = turnaround_time + end -
arrival_time[smallest];
        }
    }
    average_waiting_time = wait_time / limit;
    average_turnaround_time = turnaround_time / limit;
    printf("\n\nAverage Waiting Time:\t%lf\n", average_waiting_time);
    printf("Average Turnaround Time:\t%lf\n", average_turnaround_time);
    return 0;
}
```



```
Enter the Total Number of Processes:    4

Enter Details of 4 Processes

Enter Arrival Time:     3
Enter Burst Time:       2

Enter Arrival Time:     1
Enter Burst Time:       4

Enter Arrival Time:     3
Enter Burst Time:       5

Enter Arrival Time:     3
Enter Burst Time:       2


Average Waiting Time:   3.000000
Average Turnaround Time:        6.250000

Process returned 0 (0x0)   execution time : 9.954 s
Press any key to continue.
```