FCFS(WITHOUT AT)

```
#include<stdio.h>
struct pro{
int at,bt,pid,ct,wt,ta;
}p[10];
int n,tmp,i,j;
float ata,awt;
void read()
{
   printf("Enter number of processes ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
         p[i].ct=0;p[i].wt=0;p[i].ta=0;
         printf("Enter process id, arrival time, burst time \n");
         scanf("%d %d %d",&p[i].pid,&p[i].at,&p[i].bt);
   }
}
void sort()
{
   for(i=0;i<n-1;i++)
   {
         for(j=0;j<n-i-1;j++)
         {
                  if(p[j].at>p[j+1].at)
                  {
                           tmp=p[j].pid;
                           p[j].pid=p[j+1].pid;
                           p[j+1].pid=tmp;
                           tmp=p[j].at;
                           p[j].at=p[j+1].at;
                           p[j+1].at=tmp;
                           tmp=p[j].bt;
                           p[j].bt=p[j+1].bt;
                           p[j+1].bt=tmp;
                  }
         }
   }
}
void fcfs()
{
   p[0].ct=p[0].at+p[0].bt;
   for(i=1;i<n;i++)
   {
         if(p[i].at>p[i-1].ct)
                  p[i].ct=p[i].at+p[i].bt;
         else
                  p[i].ct=p[i-1].ct+p[i].bt;
```

```
    }
    for(i=0;i<n;i++)
    {
            p[i].ta=p[i].ct-p[i].at;
            p[i].wt=p[i].ta-p[i].bt;
    }
}
void avg()
{
    int tta=0,twt=0;
    for(i=0;i<n;i++)
    {
            tta=tta+p[i].ta;
            twt=twt+p[i].wt;
    }
    ata=(float)tta/n;
    awt=(float)twt/n;
}
void gc()
{
    printf("\n");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt;j++)
                        printf("--");
            printf(" ");
    }
    printf("\n|");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt-1;j++)
                         printf(" ");
            printf("P%d",p[i].pid);
            for(j=0;j<p[i].bt-1;j++)
                         printf(" ");
            printf("|");
    }
    printf("\n");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt;j++)
                        printf("--");
            printf(" ");
    }
    printf("\n");
    printf("0");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt;j++)
```

```
                    printf("  ");
            /*if(p[i].ta>9)
            printf("\b");*/
            printf("%d",p[i].ct);
    }
    printf("\n");
}
void display()
{
    printf("PID\tAT\tBT\tCT\tTA\tWT\n");
    for(i=0;i<n;i++)
            printf("%d\t%d\t%d\t%d\t%d\t%d\n",p[i].pid,p[i].at,p[i].bt,p[i].ct,p[i].ta,p[i].wt);
    printf("Average tunaround time %f\n",ata);
    printf("Average waiting time %f\n",awt);
}
void main()
{
    read();
    sort();
    fcfs();
    avg();
    gc();
    display();
}

/*OUTPUT
Enter number of processes 3
Enter process id, arrival time, burst time
1 0 5
Enter process id, arrival time, burst time
2 0 8
Enter process id, arrival time, burst time
3 0 12

---------- ---------------- ------------------------
|        P1       |        P2       |        P3       |
---------- ---------------- ------------------------
0        5                13                        25
PID   AT   BT   CT   TA   WT
1    0    5    5    5    0
2    0    8    13    13    5
3    0    12    25    25    13
Average tunaround time 14.333333
Average waiting time 6.000000
*/
```

FCFS(WITH ARRIVAL TIME)

```
#include<stdio.h>
struct pro{
int at,bt,pid,ct,wt,ta;
}p[10],g[100],tmp;
int n,i,j,k=0,n1;
float ata,awt;
void read()
{
    printf("Enter number of processes ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
          p[i].ct=0;p[i].wt=0;p[i].ta=0;
          printf("Enter process id, arrival time, burst time \n");
          scanf("%d %d %d",&p[i].pid,&p[i].at,&p[i].bt);
    }
}
void sort()
{
    for(i=0;i<n-1;i++)
    {
          for(j=0;j<n-i-1;j++)
          {
                    if(p[j].at>p[j+1].at)
                    {
                              tmp=p[j];
                              p[j]=p[j+1];
                              p[j+1]=tmp;
                    }
          }
    }
}
void fcfs()
{
    if(p[0].at!=0)
   {
         g[k].pid=0;
         g[k].bt=p[0].at;
         g[k].ct=p[0].at;
```

```
        k++;
    }
     p[0].ct=p[0].at+p[0].bt;
     g[k].pid=p[0].pid;
     g[k].ct=p[0].ct;
     g[k].bt=p[0].bt;
     k++;



     for(i=1;i<n;i++)
     {
          if(p[i].at>p[i-1].ct)
          {
               g[k].pid=0;
               g[k].bt=p[i].at-p[i-1].ct;
               g[k].ct=p[i].at;
               k++;
               p[i].ct=p[i].at+p[i].bt;
          }
          else
               p[i].ct=p[i-1].ct+p[i].bt;
          g[k].pid=p[i].pid;
          g[k].ct=p[i].ct;
          g[k].bt=p[i].bt;
          k++;
     }
     for(i=0;i<n;i++)
     {
          p[i].ta=p[i].ct-p[i].at;
          p[i].wt=p[i].ta-p[i].bt;
     }
     n1=k;
}
void avg()
{
     int tta=0,twt=0;
     for(i=0;i<n;i++)
     {
         tta=tta+p[i].ta;
         twt=twt+p[i].wt;
   }
   ata=(float)tta/n;
   awt=(float)twt/n;
}
void gc()
{
        printf("\nGantt Chart\n");
        int i, j;
        printf(" ");
        for(i=0; i<n1; i++)
        {
        for(j=0; j<g[i].bt; j++)
          printf("--");
        printf(" ");
        }
        printf("\n|");

        for(i=0; i<n1; i++)
        {
        for(j=0; j<g[i].bt - 1; j++)
         printf(" ");
        printf("P%d", g[i].pid);
        for(j=0; j<g[i].bt - 1; j++)
         printf(" ");
        printf("|");
        }
        printf("\n ");
        for(i=0; i<n1; i++)
        {
        for(j=0; j<g[i].bt; j++)
         printf("--");
        printf(" ");
        }
        printf("\n");
        printf("0");
        for(i=0; i<n1; i++)
        {
```

```
        for(j=0; j<g[i].bt; j++)
         printf("  ");
        if(g[i].ct > 9)
         printf("\b");
        printf("%d", g[i].ct);
        }
        printf("\n");
}
void display()
{
   printf("PID\tAT\tBT\tCT\tTA\tWT\n");
   for(i=0;i<n;i++)
         printf("%d\t%d\t%d\t%d\t%d\t%d\n",p[i].pid,p[i].at,p[i].bt,p[i].ct,p[i].ta,p[i].wt);
   printf("Average tunaround time %f\n",ata);
   printf("Average waiting time %f\n",awt);
}
void main()
{
   read();
   sort();
   fcfs();
   avg();
   gc();
   display();
}
```

SJF(WITHOUT AT)

```
#include<stdio.h>
struct pro{
int at,bt,pid,ct,wt,ta;
}p[10];
int n,tmp,i,j;
float ata,awt;
void read()
{
   printf("Enter number of processes ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
         p[i].ct=0;p[i].wt=0;p[i].ta=0,p[i].at=0;
         printf("Enter process id, burst time \n");
```

```
            scanf("%d %d",&p[i].pid,&p[i].bt);
    }
}
void sort()
{
    for(i=0;i<n-1;i++)
    {
            for(j=0;j<n-i-1;j++)
            {
                    if(p[j].bt>p[j+1].bt)
                    {
                            tmp=p[j].pid;
                            p[j].pid=p[j+1].pid;
                            p[j+1].pid=tmp;
                            tmp=p[j].bt;
                            p[j].bt=p[j+1].bt;
                            p[j+1].bt=tmp;
                    }
            }
    }
}
void sjf()
{
    p[0].ct=p[0].at+p[0].bt;
    for(i=1;i<n;i++)
            p[i].ct=p[i-1].ct+p[i].bt;
    for(i=0;i<n;i++)
    {
            p[i].ta=p[i].ct-p[i].at;
            p[i].wt=p[i].ta-p[i].bt;
    }
}
void avg()
{
    int tta=0,twt=0;
    for(i=0;i<n;i++)
    {
            tta=tta+p[i].ta;
            twt=twt+p[i].wt;
    }
    ata=(float)tta/n;
    awt=(float)twt/n;
}
void gc()
{
    printf("\n");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt;j++)
```

```
                printf("--");
            printf(" ");
    }
    printf("\n|");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt-1;j++)
                    printf(" ");
            printf("P%d",p[i].pid);
            for(j=0;j<p[i].bt-1;j++)
                    printf(" ");
            printf("|");
    }
    printf("\n");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt;j++)
                printf("--");
            printf(" ");
    }
    printf("\n");
    printf("0");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt;j++)
                    printf("  ");
            printf("%d",p[i].ct);
    }
    printf("\n");
}
void display()
{
    printf("PID\tAT\tBT\tCT\tTA\tWT\n");
    for(i=0;i<n;i++)
            printf("%d\t%d\t%d\t%d\t%d\t%d\n",p[i].pid,p[i].at,p[i].bt,p[i].ct,p[i].ta,p[i].wt);
    printf("Average tunaround time %f\n",ata);
    printf("Average waiting time %f\n",awt);
}
void main()
{
    read();
    sort();
    sjf();
    avg();
    gc();
    display();
}
```

```
/*OUTPUT
Enter number of processes 5
Enter process id, burst time
1 4
Enter process id, burst time
2 3
Enter process id, burst time
3 7
Enter process id, burst time
4 1
Enter process id, burst time
5 2

-- ---- ------ -------- --------------
|P4| P5 |  P2  |   P1   |          P3       |
-- ---- ------ -------- ----- ---------
0  1     3        6        10                  17
PID    AT    BT    CT    TA    WT
4    0    1    1    1    0
5    0    2    3    3    1
2    0    3    6    6    3
1    0    4    10    10    6
3    0    7    17    17    10
Average tunaround time 7.400000
Average waiting time 4.000000*/
```

SJF WITH AT

```
#include<stdio.h>
struct process{
   int id,at,bt,ct,ta,wt,exec;
}p[10],g[100],tmp;
int n,i,j,k=0,tta=0,twt=0;
void read()
{
   printf("input number of processes:");
   scanf("%d",&n);
   printf("Enter id,at and bt\n");
   for(i=0;i<n;i++)
   {
      scanf("%d %d %d",&p[i].id,&p[i].at,&p[i].bt);
      p[i].ct=0;
      p[i].ta=0;
      p[i].wt=0;
      p[i].exec=0;
   }
}
void sort()
```

```
{
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(p[j].at>p[j+1].at)
            {
                tmp=p[j];
                p[j]=p[j+1];
                p[j+1]=tmp;
            }
        }
    }
}
void sjf()
{
    int lt=0,rem=n,k,min;
    if(p[0].at!=0)
    {
        g[k].id=0;
        g[k].bt=p[0].at;
        g[k].ct=p[0].at;
        k++;
    }
    lt=p[0].at+p[0].bt;
    p[0].ct=lt;
    g[k++]=p[0];
    p[0].exec=1;
    rem--;
    while(rem>0)
    {

        for(i=0;i<n;i++)
            if(p[i].exec==0 && p[i].at<=lt)
            {

                break;
            }
                min=i;
            for(j=0;j<n;j++)
                if(p[j].exec==0 && p[j].at<=lt && p[j].bt<p[min].bt)
                    min=j;

        lt=lt+p[min].bt;
        p[min].ct=lt;
        p[min].exec=1;
        g[k]=p[min];+
        k++;
        rem--;
    }
    for(i=0;i<n;i++)
    {
        p[i].ta=p[i].ct-p[i].at;
        tta=tta+p[i].ta;
        p[i].wt=p[i].ta-p[i].bt;
        twt=twt+p[i].wt;
    }
}
void display()
{
    printf("ID\tAT\tBT\tCT\tTA\tWT\n");
    for(i=0;i<n;i++)
        printf("%d\t%d\t%d\t%d\t%d\t%d\n",p[i].id,p[i].at,p[i].bt,p[i].ct,p[i].ta,p[i].wt);
    printf("\n");
}
void gc()
{
    for(i=0;i<k;i++)
    {
        for(j=0;j<g[i].bt;j++)
            printf("--");
        printf(" ");
    }
    printf("\n|");
    for(i=0;i<k;i++)
    {
        for(j=0;j<g[i].bt-1;j++)
            printf(" ");
        printf("P%d",g[i].id);
        for(j=0;j<g[i].bt-1;j++)
            printf(" ");
        printf("|");
    }
    printf("\n");
    for(i=0;i<k;i++)
    {
        for(j=0;j<g[i].bt;j++)
            printf("--");
        printf(" ");
    }
    printf("\n0");
    for(i=0;i<k;i++)
    {
        for(j=0;j<g[i].bt;j++)
            printf("  ");
         if(g[i].ct>9)
            printf("\b");
        printf("%d",g[i].ct);
    }
    printf("\n");
}
void avg()
{
    float ata=(float)tta/n;
    float awt=(float)twt/n;
    printf("Average turnaround time=%f\n",ata);
    printf("Average waiting time=%f\n",awt);
}
void main()
{
   read();
   sort();
   sjf();
   display();
   gc();
   avg();
}
```

PRIORITY(WITHOUT AT)

```
#include<stdio.h>
struct pro{
int at,bt,pid,ct,wt,ta,prio;
}p[10];
int n,tmp,i,j;
float ata,awt;
void read()
{
    printf("Enter number of processes ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
            p[i].ct=0;p[i].wt=0;p[i].ta=0,p[i].at=0;
            printf("Enter process id, burst time, priority\n");
            scanf("%d %d %d",&p[i].pid,&p[i].bt,&p[i].prio);
    }
}
void sort()
{
    for(i=0;i<n-1;i++)
    {
            for(j=0;j<n-i-1;j++)
            {
                    if(p[j].prio>p[j+1].prio)
                    {
                            tmp=p[j].pid;
                            p[j].pid=p[j+1].pid;
```

```
                        p[j+1].pid=tmp;
                        tmp=p[j].bt;
                        p[j].bt=p[j+1].bt;
                        p[j+1].bt=tmp;
                        tmp=p[j].prio;
                        p[j].prio=p[j+1].prio;
                        p[j+1].prio=tmp;
                }
            }
    }
}
void prio()
{
    p[0].ct=p[0].at+p[0].bt;
    for(i=1;i<n;i++)
            p[i].ct=p[i-1].ct+p[i].bt;
    for(i=0;i<n;i++)
    {
            p[i].ta=p[i].ct-p[i].at;
            p[i].wt=p[i].ta-p[i].bt;
    }
}
void avg()
{
    int tta=0,twt=0;
    for(i=0;i<n;i++)
    {
            tta=tta+p[i].ta;
            twt=twt+p[i].wt;
    }
    ata=(float)tta/n;
    awt=(float)twt/n;
}
void gc()
{
    printf("\n");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt;j++)
                    printf("--");

            printf(" ");
    }
    printf("\n|");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt-1;j++)
                     printf(" ");
            printf("P%d",p[i].pid);
            for(j=0;j<p[i].bt-1;j++)
                        printf(" ");        printf("|");
    }
    printf("\n");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt;j++)
                        printf("--");
            printf(" ");
    }
    printf("\n");
    printf("0");
    for(i=0;i<n;i++)
    {
            for(j=0;j<p[i].bt;j++)
                        printf("  ");
            printf("%d",p[i].ct);
    }
    printf("\n");
}
void display()
{
    printf("PID\tPriority\tAT\tBT\tCT\tTA\tWT\n");
    for(i=0;i<n;i++)
            printf("%d\t%d\t\t%d\t%d\t%d\t%d\t%d\n",p[i].pid,p[i].prio,p[i].at,p[i].bt,p[i].ct,p[i].ta,p[i].wt);
    printf("Average tunaround time %f\n",ata);
    printf("Average waiting time %f\n",awt);
}
void main()
{
    read();
    sort();
    prio();
    avg();
    gc();
    display();
}
```

PRIORITY(WITH AT)

```
#include<stdio.h>
struct pro{
int at,bt,pid,ct,wt,ta,exe,prio;
}p[10],g[100],tmp;
int n,i,j,k=0,n1;
float ata,awt;
void read()
{
    printf("Enter number of processes ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
            p[i].ct=0;p[i].wt=0;p[i].ta=0,p[i].exe=0;
            printf("Enter process id, burst time, arrival time, priority \n");
            scanf("%d %d %d %d",&p[i].pid,&p[i].bt,&p[i].at,&p[i].prio);
    }
}
void sort()
{
    for(i=0;i<n-1;i++)
    {
            for(j=0;j<n-i-1;j++)
            {
                    if(p[j].at>p[j+1].at)
                    {
                            tmp=p[j];
                            p[j]=p[j+1];
                            p[j+1]=tmp;
                    }
            }
    }
}
void priority()
{
    int rem=n,lt=0;
    lt=lt+p[0].bt;
    p[0].exe=1;
    p[0].ct=lt;
    g[k]=p[0];
    k++;
    rem--;
    while(rem!=0)
    {
            for(j=1;j<n;j++)
                    if(p[j].exe!=1 && p[j].at<=lt)
                            break;
            int min=j;
            for(i=1;i<n;i++)
                    if(p[i].exe!=1 && p[i].at<=lt && p[i].prio<p[min].prio)
                            min=i;
            lt=lt+p[min].bt;
            p[min].exe=1;
            p[min].ct=lt;
            g[k]=p[min];
            k++;
            rem--;
    }
    for(i=0;i<n;i++)
    {
             p[i].ta=p[i].ct-p[i].at;
             p[i].wt=p[i].ta-p[i].bt;
    }
    n1=k;
}
void avg()
{
    int tta=0,twt=0;
    for(i=0;i<n;i++)
    {
             tta=tta+p[i].ta;
             twt=twt+p[i].wt;
    }
    ata=(float)tta/n;
    awt=(float)twt/n;
}
void gc()
{
            printf("\nGantt Chart\n");
            int i, j;
            printf(" ");
            for(i=0; i<n1; i++)
            {
            for(j=0; j<g[i].bt; j++)
             printf("--");
            printf(" ");
            }
            printf("\n|");

            for(i=0; i<n1; i++)
            {
            for(j=0; j<g[i].bt - 1; j++)
             printf(" ");
            printf("P%d", g[i].pid);
            for(j=0; j<g[i].bt - 1; j++)
             printf(" ");
            printf("|");
            }
            printf("\n ");
            for(i=0; i<n1; i++)
            {
            for(j=0; j<g[i].bt; j++)
             printf("--");
            printf(" ");
            }
            printf("\n");
            printf("0");
            for(i=0; i<n1; i++)
            {
            for(j=0; j<g[i].bt; j++)
             printf("  ");
            if(g[i].ct > 9)
             printf("\b");
            printf("%d", g[i].ct);
            }
            printf("\n");
}
void display()
{
    printf("PID\tAT\tBT\tCT\tTA\tWT\n");
    for(i=0;i<n;i++)
             printf("%d\t%d\t%d\t%d\t%d\t%d\n",p[i].pid,p[i].at,p[i].bt,p[i].ct,p[i].ta,p[i].wt);
    printf("Average tunaround time %f\n",ata);
    printf("Average waiting time %f\n",awt);
}
void main()
{
    read();
    sort();
    priority();
    avg();
    gc();
    display();
}
```

<u>ROUND ROBIN</u>

```
#include<stdio.h>
typedef struct
{ int id, at, bt, ct ,ta,wt,exe,rt; }pro;
pro p[10],g[100],temp;
int n,n1,q[10],front=-1,rear=-1,ts;
void read()
{
            printf("enter the no. of process:");
            scanf("%d",&n);
            printf("Enter the time quanta : ");
            scanf("%d",&ts);
            for(int i=0;i<n;i++)
            {
                        p[i].ct=0,p[i].ta=0;p[i].wt=0,p[i].exe=0;
                        printf("enter the id,at,bt of the process:");
                        scanf("%d%d%d",&p[i].id,&p[i].at,&p[i].bt);
                        p[i].rt=p[i].bt;
            }
}
            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)
            {
                temp=p[j];
                p[j]=p[j+1];
                p[j+1]=temp;
            }
        }
    }
}
void sort1()
{
    for(int i=0;i<n-1;i++)
    {
        for(int j=0;j<n-i-1;j++)
        {
            if(p[j].id>p[j+1].id)
            {
                temp=p[j];
                p[j]=p[j+1];
                p[j+1]=temp;
            }
        }
    }
}
void display()
{
    printf("\npid\tat\tbt\tct\tta\twt\n");
    for(int i=0;i<n;i++)
    {
        printf("p%d\t%d\t%d\t%d\t%d\t%d\n",p[i].id,p[i].at,p[i].bt,p[i].ct,p[i].ta,p[i].wt);
      ● }
}
void enqueue(int item)
{
    rear=(rear+1)%n;
    q[rear]=item;
    if(front==-1)
        front=0;
}
int dequeue()
{

    int item=q[front];
    if(front==rear)
    {
        front=-1;
```

```
                rear=-1;
        }
        else
                front=(front+1)%n;
        return item;
}
void rr()
{
        int remain=n,prorem=n;
        enqueue(p[0].id);
        p[0].exe=1;
        prorem--;
        int lt=0,i,k=0;
        while(remain!=0)
        {
                int flag=0;
                int qpid=dequeue();
                for(i=0;i<n;i++)
                        if(p[i].id==qpid)
                                break;
                if(p[i].rt<=ts)
                {
                        int tempbt=p[i].rt;
                        p[i].ct=lt+p[i].rt;
                        lt=lt+p[i].rt;
                        p[i].rt=0;
                        g[k].id=p[i].id;
                        g[k].bt=tempbt;
                        g[k].ct=lt;
                        k++;
                        flag=1;
                        remain--;
                }
                else
                {
                        p[i].rt=p[i].rt-ts;
                        lt=lt+ts;
                        g[k].id=p[i].id;
                        g[k].bt=ts;
                        g[k].ct=lt;
                        k++;
                }
                if(prorem!=0)
                {
                        for(int j=0;j<n;j++)
                                if(p[j].exe!=1 && p[j].at<=lt)
                                {
                                        enqueue(p[j].id);
                                        p[j].exe=1;
```

```
                                                prorem--;
                                        }
                        }
                        if(flag!=1)
                                enqueue(p[i].id);
                }
                for(int i=0;i<n;i++)
                {
                        p[i].ta=p[i].ct-p[i].at;
                        p[i].wt=p[i].ta-p[i].bt;
                }
                n1=k;
}
void gc()
{
    printf("\nThe Gantt Chart\n");
    int i, j;
    printf(" ");
    for(i=0; i<n1; i++)
    {
        for(j=0; j<g[i].bt; j++)
             printf("--");
        printf(" ");
    }
    printf("\n|");

    for(i=0; i<n1; i++)
    {
        for(j=0; j<g[i].bt - 1; j++) printf(" ");
        printf("P%d", g[i].id);
        for(j=0; j<g[i].bt - 1; j++) printf(" ");
        printf("|");
    }
    printf("\n ");
    for(i=0; i<n1; i++)
    {
        for(j=0; j<g[i].bt; j++) printf("--");
        printf(" ");
    }
    printf("\n");
    printf("0");
    for(i=0; i<n1; i++)
    {
        for(j=0; j<g[i].bt; j++) printf("  ");
        if(g[i].ct > 9)
            printf("\b");
        printf("%d", g[i].ct);

    }
    printf("\n");
}
void avg()
{
                float sumta=0,sumwt=0;
                float avgta,avgwt;
                for(int i=0;i<n;i++)
                {
                        sumta=p[i].ta+sumta;
                        sumwt=p[i].wt+sumwt;
                }
                avgta=sumta/n;
                avgwt=sumwt/n;
                printf("\nAverage TurnAroundTime = %.2f",avgta);
                printf("\nAverage WaitingTime = %.2f",avgwt);
}
int main()
{
                read();
                sort();
                rr();
                sort1();
                display();
                gc();
                avg();
}
```

PRODUCER CONSUMER

```
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>
#define MaxItems 5
#define BufferSize 5

sem_t empty;
sem_t full;
int in = 0;
int out= 0;
    ●   int buffer[BufferSize];
pthread_mutex_t mutex;

void *producer(void *pno)
{
        int item;
        for(int i = 0; i < MaxItems; i++) {
        item = rand();
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        }
}
void *consumer(void *cno)
{
        for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item, out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        }
}

int main()
{

        pthread_t pro[5],con[5];
        pthread_mutex_init(&mutex, NULL);
        sem_init(&empty,0,BufferSize);
        sem_init(&full,0,0);

        int a[5] = {1,2,3,4,5};
        for(int i = 0; i < 5; i++) {
                pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
        }
        for(int i = 0; i < 5; i++) {
                pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
        }

        for(int i = 0; i < 5; i++) {
                pthread_join(pro[i], NULL);
        }
        for(int i = 0; i < 5; i++) {
        pthread_join(con[i], NULL);

        }

        pthread_mutex_destroy(&mutex);
        sem_destroy(&empty);
        sem_destroy(&full);
        return 0;
```

```
}
```

DINING PHILOSOPHER

```
#include<stdio.h>
#include<semaphore.h>
#include<pthread.h>
#define N 5
#define LEFT (i+N-1)%N
#define RIGHT (i)%N
#define THINKING 0
#define HUNGRY 1
#define EATING 2
int state[N];
pthread_t t[N];
sem_t s[N];
sem_t mutex;

void think(int n)
{
printf("The philosopher %d is thinking\n",n);
sleep(1);
}
void eat(int n)
{
printf("\t\t\tThe philosopher %d is eating\n",n);
sleep(1);
printf("\t\t\tThe philosopher %d has finished eating\n",n);
}
void take_forks(int i)
{
sem_wait(&mutex);
state[i]=HUNGRY;
if(state[i]==HUNGRY && state[LEFT]!=EATING && state[RIGHT]!=EATING)
{
state[i]=EATING;
sem_wait(&s[LEFT]);
sem_wait(&s[RIGHT]);
}
sem_post(&mutex);
}
void put_forks(int i)
{
state[i]=THINKING;
sem_post(&s[LEFT]);
sem_post(&s[RIGHT]);
}
void *philo(int n)
{
while(1)
{
think(n);
take_forks(n);
if(state[n]==EATING)
{
eat(n);
put_forks(n);
}
}
}

main()
{
int i;
for(i=0;i<N;i++)
{
sem_init(&s[i],0,1);
}
sem_init(&mutex,0,1);
for(i=0;i<N;i++)
{
pthread_create(&t[i],0,(void *)philo,(void *)i);
}
while(1);
}
```

MEMORY ALLOCATION

```
#include<stdio.h>

struct process{
int pid,psize,pstatus,d;
}p[100];
struct block{
int bid,bsize,bstatus,alloc;
}b[100],temp;
int n,m;
int i,j,waste=0;
void firstfit()
{
    for(i=0;i<n;i++)
    {
            for(j=0;j<m;j++)
            {
```

```
                        if((b[j].bsize>=p[i].psize)&&(b[j].bstatus!=1))
                        {
                                b[j].bstatus=1;
                                b[j].alloc=p[i].pid;
                                break;
                        }
                }
        }
        printf("Firstfit\n");
        printf("Process\tProcesssize\tBlock\tBlocksize\tWastage\n");
        for(i=0;i<m;i++)
        {
                if(b[i].alloc!=-1)
                {
                        int k=b[i].alloc-1;
                        waste=waste+b[i].bsize-p[k].psize;
                        printf("%d\t%d\t\t%d\t%d\t\t%d\n",p[k].pid,p[k].psize,b[i].bid,b[i].bsize,(b[i].bsize-p[k].psize));
                        p[k].d=1;
                }
        }
        for(i=0;i<n;i++)
        {
                if(p[i].d==0)
                printf("%d\t%d\t\t-\t-\t\t-\n",p[i].pid,p[i].psize);
        }
}
void sort1()
{
        for(i=0;i<m-1;i++)
        {
                for(j=0;j<m-i-1;j++)
                {
                        if(b[j].bsize>=b[j+1].bsize)
                        {
                                temp=b[j];
                                b[j]=b[j+1];
                                b[j+1]=temp;
                        }
                }
        }
}
void sort2()
{
        for(i=0;i<m-1;i++)
        {
                for(j=0;j<m-i-1;j++)
                {
                        if(b[j].bsize<=b[j+1].bsize)
                        {
```

```
                        temp=b[j];
                        b[j]=b[j+1];
                        b[j+1]=temp;
                }
            }
    }
}
void bestfit()
{
    sort1();
    for(i=0;i<n;i++)
    {
            for(j=0;j<m;j++)
            {
                    if((b[j].bsize>=p[i].psize)&&(b[j].bstatus!=1))
                    {
                            b[j].bstatus=1;
                            b[j].alloc=p[i].pid;
                            break;
                    }
            }
    }
    printf("Bestfit\n");
    printf("Process\tProcesssize\tBlock\tBlocksize\tWastage\n");
    for(i=0;i<m;i++)
    {
            if(b[i].alloc!=-1)
            {
                    int k=b[i].alloc-1;
                    waste=waste+b[i].bsize-p[k].psize;
                    printf("%d\t%d\t\t%d\t%d\t\t%d\n",p[k].pid,p[k].psize,b[i].bid,b[i].bsize,(b[i].bsize-p[k].psize));
                    p[k].d=1;
            }
    }
    for(i=0;i<n;i++)
    {
            if(p[i].d==0)
            printf("%d\t%d\t\t-\t-\t\t-\n",p[i].pid,p[i].psize);
    }
}
void worstfit()
{
    sort2();
    for(i=0;i<n;i++)
    {
            for(j=0;j<m;j++)
            {
                    if((b[j].bsize>=p[i].psize)&&(b[j].bstatus!=1))
                    {
```

```
                                b[j].bstatus=1;
                                b[j].alloc=p[i].pid;
                                break;
                        }
                }
        }
        printf("Worstfit\n");
        printf("Process\tProcesssize\tBlock\tBlocksize\tWastage\n");
        for(i=0;i<m;i++)
        {
                if(b[i].alloc!=-1)
                {
                        int k=b[i].alloc-1;
                        waste=waste+b[i].bsize-p[k].psize;
                        printf("%d\t%d\t\t%d\t%d\t\t%d\n",p[k].pid,p[k].psize,b[i].bid,b[i].bsize,(b[i].bsize-p[k].psize));
                        p[k].d=1;
                }
        }
        for(i=0;i<n;i++)
        {
                if(p[i].d==0)
                printf("%d\t%d\t\t-\t-\t\t-\n",p[i].pid,p[i].psize);
        }
}
void main()
{

        printf("input number of processes:");
        scanf("%d",&n);
        printf("input process size of each process\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&p[i].psize);
                p[i].pid=i+1;
                p[i].pstatus=0;
                p[i].d=0;
        }
        printf("input number of blocks:");
        scanf("%d",&m);
        printf("input process size of each blocks\n");
        for(i=0;i<m;i++)
        {
                scanf("%d",&b[i].bsize);
                b[i].bid=i+1;
                b[i].bstatus=0;
                b[i].alloc=-1;
        }
        firstfit();
        printf("total wastage=%d\n",waste);
```

```
    for(i=0;i<n;i++)
    {
            p[i].pstatus=0;
            p[i].d=0;
    }
    for(i=0;i<m;i++)
    {
            b[i].bstatus=0;
            b[i].alloc=-1;
    }
    waste=0;
    bestfit();
    printf("total wastage=%d\n",waste);
    waste=0;
    for(i=0;i<n;i++)
    {
            p[i].pstatus=0;
            p[i].d=0;
    }
    for(i=0;i<m;i++)
    {
            b[i].bstatus=0;
            b[i].alloc=-1;
    }
    worstfit();
    printf("total wastage=%d\n",waste);
}
```

BANKERS ALGORITHM

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

int avail[100];
int resmax[100];
int maxalloc[100];
int req[100];
struct process
{
  char name[100];
  int max[100];
  int alloc[100];
  int need[100];
  int done;
}p[20],temp;
```

```
void main()
{
  int i,j,r,pr,flag,ls,ml=0,g=0,id;
  char name[100],str[100] = "";
  printf("ENTER THE NUMBER OF RESOURCES : ");
  scanf("%d",&r);
  printf("MAXIMUM RESOURCE COUNT FOR : \n");
  for(j=0;j<r;j++)
  {
    printf("\tRESOURCE %d : ",j+1);
    scanf("%d",&resmax[j]);
  }
  printf("\nENTER THE NUMBER OF PROCESSES : ");
  scanf("%d",&pr);
  for(i=0;i<pr;i++)
  {
    printf("\nENTER PROCESS NAME : ");
    scanf(" %s",p[i].name);
    printf("\nDETAILS FOR PROCESS %s",p[i].name);
    for(j=0;j<r;j++)
    {
      printf("\n\tMAXIMUM ALLOCATION FOR RESOURCE %d : ",j+1);
      scanf("%d",&p[i].max[j]);

      printf("\tALLOCATED RESOURCE FOR RESOURCE %d : ",j+1);
      scanf("%d",&p[i].alloc[j]);

      maxalloc[j] = maxalloc[j] + p[i].alloc[j];
      p[i].need[j] = p[i].max[j] - p[i].alloc[j];
    }
    p[i].done = 0;
  }
  for (i=0;i<r;i++)
  {
    avail[i] = resmax[i] - maxalloc[i];
  }
  printf("\nENTER THE NEW REQUEST :-  \n\n");
  printf("ENTER THE PROCESS NAME : ");
  scanf("%s",name);
  for(j=0;j<r;j++)
  {
      printf("\tREQUEST FOR RESOURCE %d : ",j+1);
      scanf("%d",&req[j]);
  }
  for (i=0;i<pr;i++)
  {
      if(strcmp(p[i].name,name)==0)
      {
          id = i;
```

```
        break;
    }
}
for(flag=0,i=0;i<r;i++)
{
    if(req[i] <= p[id].need[i] && req[i]<=avail[i])
    {
        flag++;
    }
}
if(flag !=r)
{
    printf("RESOURCE NOT GRANTED!! \nREQUESTED RESOURCE GREATER THAN NEEDED");
    exit(0);
}
else if(flag == r)
{
    for(i=0;i<r;i++)
    {
        p[id].alloc[i] =  p[id].alloc[i] + req[i];
        p[id].need[i] =  p[id].need[i] - req[i];
        avail[i] = avail[i] - req[i];
    }
}
printf("\n  PROCESS \tMAXIMUM \tALLOCATED \tREMAINING\n");
for(i=0;i<pr;i++)
{
  printf("\n     %s      \t ",p[i].name);
  for(j=0;j<r;j++)    //Max
  {
    printf("%d ",p[i].max[j]);
  }
  printf(" \t  ");
  for(j=0;j<r;j++)    //Alloc
  {
    printf("%d ",p[i].alloc[j]);
  }
  printf("\t  ");
  for(j=0;j<r;j++)    //Need
  {
    printf("%d ",p[i].need[j]);
  }
}

printf("\n\nORDER OF EXECUTION :- \n");
for(i=0,ls=0;ls<pr;)
{
  for(flag = 0,j=0;j<r;j++)
  {
      if(avail[j]>=p[i].need[j])
      {
        flag++;
      }
    }
    if(flag == r && p[i].done == 0)
    {
      p[i].done = ls+1;
      for(ml=0,j=0;j<r;j++)
      {
        avail[j] = avail[j] + p[i].alloc[j];
        if(avail[j]==resmax[j])
        {
          ml++;
        }
      }
      g++;
      ls++;
      printf("\t\t%s IS VISTED \n",p[i].name);
      strcat(str,p[i].name);
      strcat(str,", ");
    }
    else
    {
      i++;
      if(i==pr)
      {
        if(g==0)
        {
          printf("\t\tREQUEST NOT ALLOCATED -- DEADLOCK OCCURED\n");
          break;
        }
        i=0;
        g=0;
      }
    }
  }
  if(ml==r && ls == pr)
  {
    printf("\nSYSTEM IS IN SAFE STATE\n");
    printf("\nSAFE STATE SEQUENCE : %s",str);
    printf("\b\b.  \n");
  }
  else if(g==0)
  {
    printf("\nSYSTEM IS IN UNSAFE STATE\n");
  }
}
```

FIFO PAGE REPLACEMENT

```
#include <stdio.h>
void main()
{
        int i,j,n,m,fnd,pg[100],fr[100],k=0,cnt=0,hit=0;
        printf("ENTER THE NUMBER OF PAGES : ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("ENTER THE PAGE NUMBER %d: ",i+1);
                scanf("%d",&pg[i]);
        }
        printf("ENTER THE NUMBER OF FRAMES : ");
        scanf("%d",&m);
        for(i=0;i<m;i++)
        {
                fr[i]=-1;
        }
        printf("\n\tREFERENCE STRING\tPAGE NUMBER\t\tSTATUS\n");
        for(i=0;i<n;i++)
        {
                fnd=0;
                printf("\t\t%d\t\t",pg[i]);
                for(j=0;j<m;j++)
                {
                if(fr[j]==pg[i])
                        {
                                fnd = 1;
                        }
                }


                if(fnd == 0)
                {
                        fr[k] = pg[i];
                        k = (k+1)%m;
                        cnt++;
                }
                for(j=0;j<m;j++)
                {
                        if(fr[j] != -1)
                        {
                                printf("%d\t",fr[j]);
                        }
                        Else
                                printf(“ \t”);
                }
                if(fnd==1)
                {
                        printf("\t\tHIT\n");
                        hit++;
                }
                else
                {
                        printf("\t\tMISS\n");
                }
        }
        printf("\nPAGE FAULT : %d\n",cnt);
        printf("\nFAULT RATIO : %d:%d\n",cnt,n);
        printf("\nNo OF HITS : %d\n",hit);
        printf("\nHIT RATIO : %d:%d\n",hit,n);
}
```

LRU PAGE REPLACEMENT

```
#include <stdio.h>
void main()
{
        int i,j,n,m,fnd,pg[100],fr[100],k=0,cnt=0,hit=0;
        printf("ENTER THE NUMBER OF PAGES : ");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("ENTER THE PAGE NUMBER %d: ",i+1);
                scanf("%d",&pg[i]);
        }
        printf("ENTER THE NUMBER OF FRAMES : ");
        scanf("%d",&m);
        for(i=0;i<m;i++)
        {
                fr[i]=-1;
        }
        printf("\n\tREFERENCE STRING\tPAGE NUMBER\tSTATUS\n");
        for(i=0;i<n;i++)
        {
                fnd=0;
                printf("\t\t%d\t\t",pg[i]);
                for(j=0;j<m;j++)
                {
                        if(fr[j]==pg[i])
                        {
                                fnd = 1;
                        }
                }
                if(fnd == 0)
                {
                        fr[k] = pg[i];
                        k = (k+1)%m;
                        cnt++;
                }
                for(j=0;j<m;j++)
                {
                        if(fr[j] != -1)
                        {
                                printf("%d\t",fr[j]);
                        }
                        else
                           printf(" \t");
                }
                if(fnd==1)
                {
                        printf("HIT\n");
                        if(fr[k]==pg[i])
                        {
                             k = (k+1)%m;
                        }
                        hit++;
                }
                else
                {
                        printf("MISS\n");
                }
        }
        printf("\nPAGE FAULT : %d\n",cnt);
        printf("\nFAULT RATIO : %d:%d\n",cnt,n);
        printf("\nNo OF HITS : %d\n",hit);
        printf("\nHIT RATIO : %d:%d\n",hit,n);
}
```

LFU PAGE REPLACEMENT

```
include<stdio.h>
struct frame{
 int content,count,cnt;
 }fr[100];
int main()
{
 int i,j,pg[100],fnd,hit=0,n,m,pf=
0,min=0,id=0,k,cnt=1;
 printf("enter no of pages:");
 scanf("%d",&n);
 printf("enter string:\n");
 for(i=0;i<n;i++)
 {
  scanf("%d",&pg[i]);
 }
 printf("enter no of frames:");
```

```
scanf("%d",&m);
for(i=0;i<m;i++)
{

  fr[i].content=-1;
  fr[i].count=0;
  fr[i].cnt=0;
}
  printf("\nreferencing_page\tstatus\t\t content\n");
 for(i=0;i<n;i++)
 {
   printf("\t%d\t\t",pg[i]);
   for(j=0;j<m;j++)
   {
    if(fr[j].content==pg[i])
    {
      printf("HIT\t\t");
      fr[j].count++;
      hit++;
      break;
    }
  }
  if(j==m)
  {
   printf("MISS\t\t");
   if(id<m)
   {
     fr[id].content=pg[i];
     fr[id].count++;
     fr[id].cnt=cnt++;
     id++;
   }
   else
   {
    for(j=0;j<m;j++)
    {
      if(fr[min].count>fr[j].count)
      {
        min=j;
      }
      else if(fr[min].count==fr[j].count && fr[j].cnt<fr[min].cnt)
      {

        min=j;
     }
    }
   fr[min].content=pg[i];
   fr[min].count=1;
   fr[min].cnt=cnt++;
   }
  pf++;
 }
 for(j=0;j<m;j++)
 {
  if(fr[j].content!=-1)
  {
    printf("%d\t",fr[j].content);
  }
 }
 printf("\n");
}
printf("\t HIT:%d\n",hit);
printf("\t MISS:%d\n",pf);
}
```

FCFS DISC SCHEDULING

```
#include<stdio.h>
#include<string.h>
void main()
{
	int tr[20],cr,n,i,sum=0,new;
	printf("ENTER THE NUMBER OF TRACKS : ");
	scanf("%d",&n);
	printf("ENTER THE HEAD POINTER POSITION : ");
	scanf("%d",&cr);
	printf("ENTER THE TRACKS TO BE TRAVERSED : ");
	for(i=0;i<n;i++)
	{
		new = 0;
		scanf("%d",&tr[i]);
		new=cr-tr[i];
		if(new<0)
		{
			new=tr[i]-cr;
		}
		cr=tr[i];
		sum=sum + new;
	}
	printf("TRAVERSED ORDER : ");
	for(i=0;i<n;i++)
	printf("%d => ",tr[i]);
	printf("\b\b. \nTOTAL HEAD MOVEMENTS : %d\n",sum);
}
```

SCAN DISC SCHEDULING

```
#include<stdio.h>
int n,m,i,j,h,p,temp,k,total=0;
int t[100],a[100],diff;
void main()
{
        printf("ENTER THE NUMBER OF TRACKS : ");
        scanf("%d",&n);
        printf("ENTER THE HEAD POINTER POSITION : ");
        scanf("%d",&h);
        printf("ENTER THE TRACKS TO BE TRAVERSED : ");
        for(i=0;i<n;i++)
        {
                scanf("%d",&t[i]);
        }
        t[n+2] = 199;
        t[n+1] = 0;
        t[n] = h;
        n=n+3;
        for(i=0;i<n;i++)
        {
                for(j=0;j<n-i-1;j++)
                {
                        if(t[j]>t[j+1])
                        {
                                temp=t[j];
                                t[j]=t[j+1];
                                t[j+1]=temp;
                        }
                }
        }
        for(i=0;i<n;i++)
        {
                if(t[i]==h)
                {
                        k=i;
                        break;
                }
        }
        /*if(h<(199-h))
        {
        for(i=k;i>=0;i--,p++)
        {
        a[p]=t[i];
        }
        for(i=k+1;i<n-1;i++,p++)
        {
        a[p]=t[i];
        }
        }
```

```
	else
	{*/
	for(i=k;i<n;i++,p++)
	{
		a[p]=t[i];
	}
	for(i=k-1;i>=0;i--,p++)
	{
		a[p]=t[i];
	}
	//}
	printf("TRAVERSED ORDER : ");
	for(i=0;i<p;i++)
	{
		printf("%d => ",a[i]);
	}
	for(total=0,j=0;j<p-1;j++)
	{
		diff=0;
		if(a[j]>a[j+1])
		{
			diff=a[j]-a[j+1];
		}
		else
		{
			diff=a[j+1]-a[j];
		}
		total=total+diff;
	}
	printf("\b\b. \nTOTAL HEAD MOVEMENTS : %d\n",total);
	printf("\b\b. \nTOTAL HEAD MOVEMENTS : %d\n",total);
}
```

C-SCAN DISC SCHEDULING

```
#include<stdio.h>
int n,m,i,j,h,p,temp,k,total=0;
int t[100],a[100],diff[100];
void main()
{
	printf("ENTER THE NUMBER OF TRACKS : ");
	scanf("%d",&n);
	printf("ENTER THE HEAD POINTER POSITION : ");
	scanf("%d",&h);
	printf("ENTER THE TRACKS TO BE TRAVERSED : ");
	for(i=0;i<n;i++)
	{
		scanf("%d",&t[i]);
}
t[n+2] = 199;
t[n+1] = 0;
t[n] = h;
n=n+3;
for(i=0;i<n;i++)
{
        for(j=0;j<n-i-1;j++)
        {
                if(t[j]>t[j+1])
                {
                        temp=t[j];
                        t[j]=t[j+1];
                        t[j+1]=temp;
                }
        }
}
for(i=0;i<n;i++)
{
        if(t[i]==h)
        {
                k=i;
                break;
        }
}
/*if(h<(199-h))
{
for(i=k;i>=0;i--,p++)
{
a[p]=t[i];
}
for(i=n-1;i>k;i--,p++)
{
a[p]=t[i];
}
}*/
//else
//{
for(i=k;i<n;i++,p++)
{
        a[p]=t[i];
}
for(i=0;i<k;i++,p++)
{
        a[p]=t[i];
}
//}
printf("TRAVERSED ORDER : ");
for(i=0;i<p;i++)
```

```
        {
                printf("%d => ",a[i]);
        }
        for(total=0,j=0;j<p-1;j++)
        {
                int diff=0;
                if(a[j]>a[j+1])
                {
                        diff=a[j]-a[j+1];
                }
                else
                {
                        diff=a[j+1]-a[j];
                }
                total=total+diff;
        }
        printf("\b\b\b. \nTOTAL HEAD MOVEMENTS : %d\n",total);
}
```