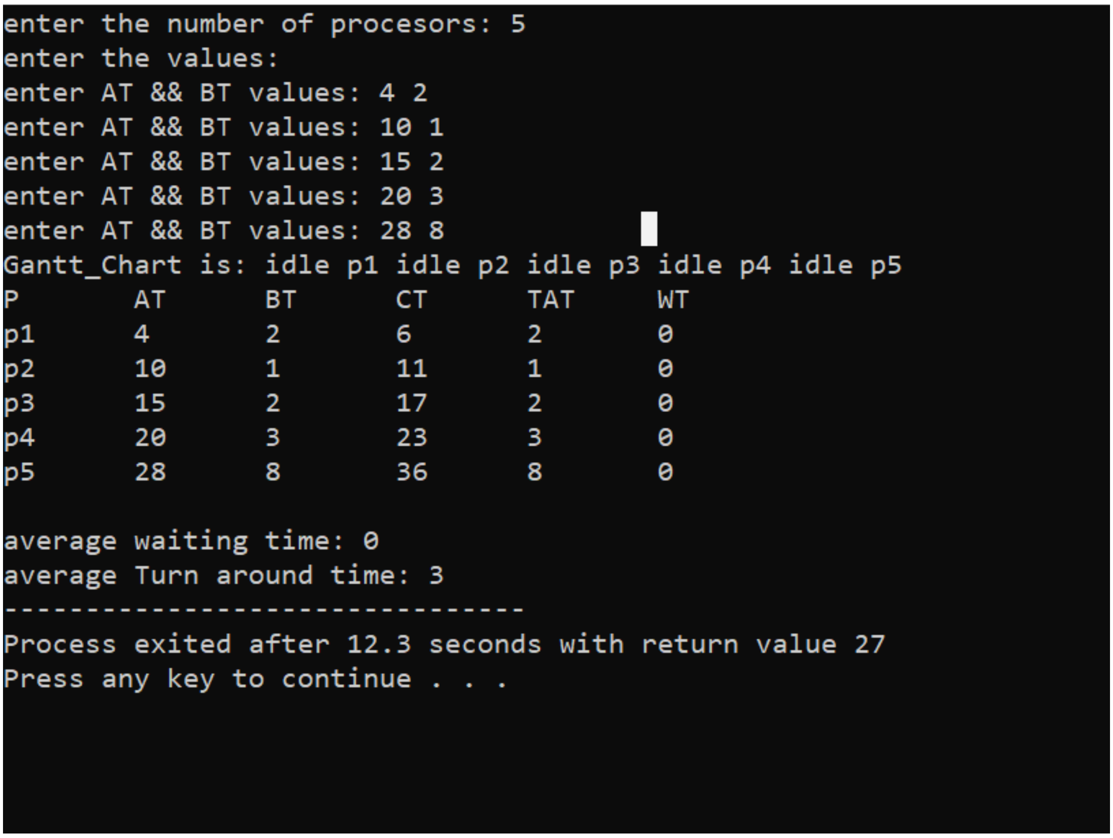
OPERATING SYSTEMS LAB

Scheduling Algorithms:

1. FCFS(FIRST COME FIRST SERVE):

```
#include<stdio.h>
int find_min(int n,int arr[][6],int i,int visit[])
{
        int min=9999,flag;
        int j;
        for(j=1;j<=n;j++)
        {
                if(arr[j][1]<min && visit[j]==-1)
                {
                        min=arr[j][i];
                        flag=j;
                }
        }
        return flag;
}
int main()
{
        int n,i,ct=0,j;
        printf("enter the number of procesors: ");
        scanf("%d",&n);
        int arr[n][6],gantt_chart[100],visit[n];
        //P     AT      BT      CT      TAT     WT

        for(i=1;i<=n;i++)
        {
                arr[i][0]=i;
                visit[i]=-1;
        }
        printf("enter the values: \n");
        for(i=1;i<=n;i++)
        {
                printf("enter AT && BT values: ");
                scanf("%d %d",&arr[i][1],&arr[i][2]);
        }
        int p=-1,l=0,k=0;
        for(i=1;i<=n;i++)
        {
                if(arr[i][1]==0)
                {
                        p=i;
                        break;
                }
        }
        if(p==-1)
        {
                gantt_chart[l]=-1;
                l++;
                p=find_min(n,arr,1,visit);
                gantt_chart[l]=p;
                l++;
                visit[p]=1;
                arr[p][3]=arr[p][1]+arr[p][2];
                ct=arr[p][3];
                k++;
        }
        else
        {
                gantt_chart[l]=p;
                l++;
                k++;
                visit[p]=1;
                arr[p][3]=arr[p][2];
                ct=arr[p][3];
        }
        while(1)
        {
                p=find_min(n,arr,1,visit);
                if(arr[p][1]>ct)
                {
                        gantt_chart[l]=-1;
                        l++;
                        ct=arr[p][1];
                }
                visit[p]=1;
                k++;
                gantt_chart[l]=p;
                l++;
                arr[p][3]=ct+arr[p][2];
                ct=arr[p][3];
                if(k==n)
                {
                        break;
                }
        }
        for(i=1;i<=n;i++)
        {
                arr[i][4]=arr[i][3]-arr[i][1];
                arr[i][5]=arr[i][4]-arr[i][2];
        }
        printf("Gantt_Chart is: ");
        for(i=0;i<l;i++)
        {
                if(gantt_chart[i]==-1)
                {
                        printf("idle ");
                }
                else
                        printf("p%d ",gantt_chart[i]);
        }
        printf("\n");
        printf("P\tAT\tBT\tCT\tTAT\tWT\n");
        for(i=1;i<=n;i++)
        {
                printf("p%d       ",i);
                for(j=1;j<6;j++)
                {
                        printf("%d\t",arr[i][j]);
                }
                printf("\n");
        }
}
```

Output:



```
enter the number of procesors: 5
enter the values:
enter AT && BT values: 4 2
enter AT && BT values: 10 1
enter AT && BT values: 15 2
enter AT && BT values: 20 3
enter AT && BT values: 28 8
Gantt_Chart is: idle p1 idle p2 idle p3 idle p4 idle p5
P       AT      BT      CT      TAT     WT
p1      4       2       6       2       0
p2      10      1       11      1       0
p3      15      2       17      2       0
p4      20      3       23      3       0
p5      28      8       36      8       0

average waiting time: 0
average Turn around time: 3
--------------------------------
Process exited after 12.3 seconds with return value 27
Press any key to continue . . .
```

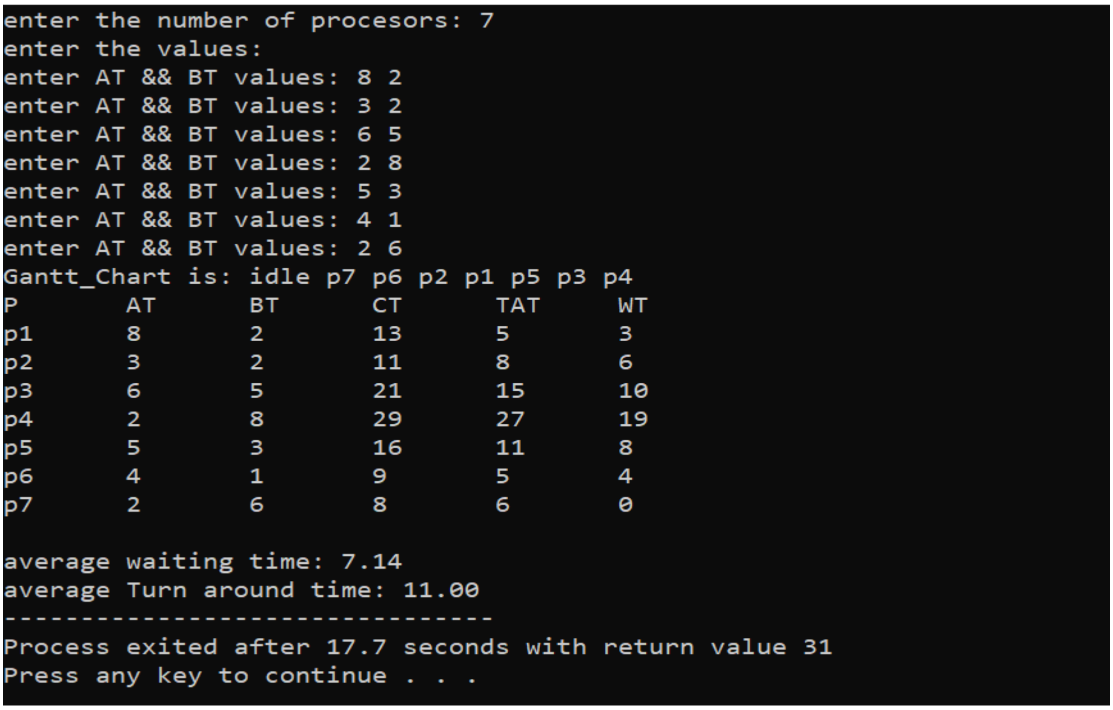
2.SJF(SHORTEST JOB FIRST)

```
#include<stdio.h>
int find_min_at(int n,int arr[][6],int i,int visit[])
{
        int min=9999,flag,ct=0;
        int j;
        for(j=1;j<=n;j++)
        {
                if(arr[j][1]<min && visit[j]==-1)
                {
                        min=arr[j][i];
                        flag=j;
                }
        }
        return flag;
}
int find_min_bt(int n,int arr[][6],int i,int visit[],int ct)
{
        int min=9999,flag=-1;
        int j;
        for(j=1;j<=n;j++)
        {
                if((arr[j][i]<min || (arr[j][i]==min && arr[j][1]<arr[flag][1])) && visit[j]==-1
&& arr[j][1]<=ct)
                {
                        min=arr[j][i];
                        flag=j;
                }
        }
        return flag;

}
int main()
{
        int n,i,ct=0,j;
        printf("enter the number of procesors: ");
        scanf("%d",&n);
        int arr[n][6],gantt_chart[100],visit[n];
        //P     AT      BT      CT      TAT     WT

        for(i=1;i<=n;i++)
        {
                arr[i][0]=i;
                visit[i]=-1;
        }
        printf("enter the values: \n");
        for(i=1;i<=n;i++)
        {
                printf("enter AT && BT values: ");
                scanf("%d %d",&arr[i][1],&arr[i][2]);
        }
```

```
    int p=-1,l=0,k=0;
    int min=999;
    for(i=1;i<=n;i++)
    {
        if(min>arr[i][1] || (min==arr[i][1] && arr[p][2]>arr[i][2]))
        {
            min=arr[i][1];
            p=i;
        }
    }
    ct=min;
    if(min!=0)
    {
        gantt_chart[l]=-1;
        l++;
        gantt_chart[l]=p;
        l++;
        visit[p]=1;
        arr[p][3]=arr[p][1]+arr[p][2];
        ct=arr[p][3];
        k++;
    }
    else
    {
        gantt_chart[l]=p;
        l++;
        k++;
        visit[p]=1;
        arr[p][3]=arr[p][2];
        ct=arr[p][3];
    }
    while(k!=n)
    {
        p=find_min_bt(n,arr,2,visit,ct);
        if(p==-1)
        {
            gantt_chart[l]=-1;
            l++;
            p=find_min_at(n,arr,1,visit);
            ct=arr[p-1][1];
            p=find_min_bt(n,arr,2,visit,ct);
        }
        visit[p]=1;
        k++;
        gantt_chart[l]=p;
        l++;
        arr[p][3]=ct+arr[p][2];
        ct=arr[p][3];
    }
    int sum_wt=0,sum_tat=0;
    for(i=1;i<=n;i++)
```

```
        {
                arr[i][4]=arr[i][3]-arr[i][1];
                arr[i][5]=arr[i][4]-arr[i][2];
                sum_wt+=arr[i][5];
                sum_tat+=arr[i][4];
        }
        printf("Gantt_Chart is: ");
        for(i=0;i<l;i++)
        {
                if(gantt_chart[i]==-1)
                {
                        printf("idle ");
                }
                else
                        printf("p%d ",gantt_chart[i]);
        }
        printf("\n");
        printf("P\tAT\tBT\tCT\tTAT\tWT\n");
        for(i=1;i<=n;i++)
        {
                printf("p%d       ",i);
                for(j=1;j<6;j++)
                {
                        printf("%d\t",arr[i][j]);
                }
                printf("\n");
        }

        printf("\naverage waiting time: %d\n",sum_wt/n);
        printf("average Turn around time: %d",sum_tat/n);
}
```

Output:



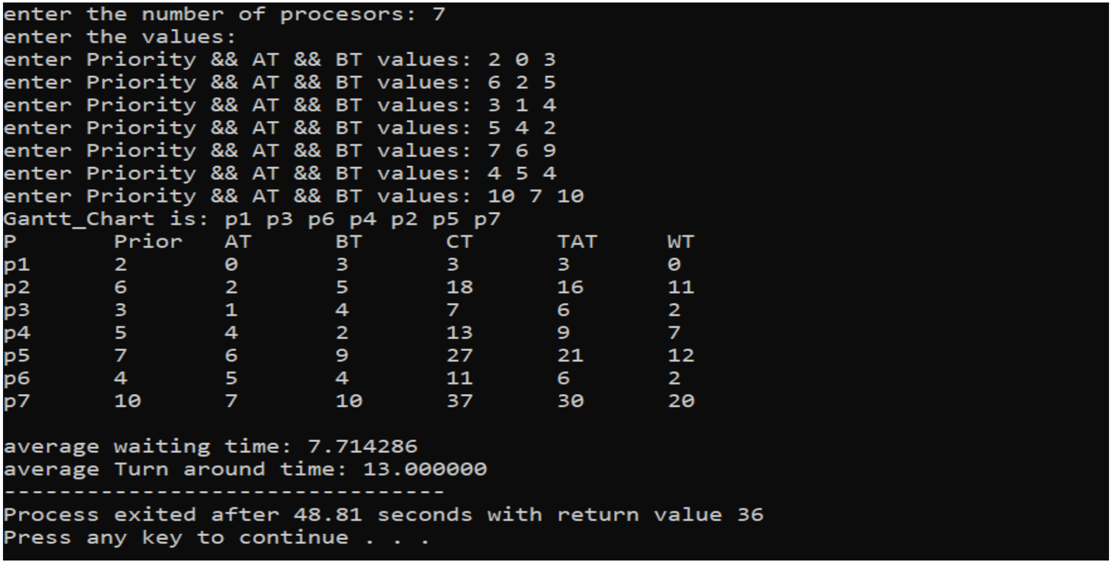
```
enter the number of procesors: 7
enter the values:
enter AT && BT values: 8 2
enter AT && BT values: 3 2
enter AT && BT values: 6 5
enter AT && BT values: 2 8
enter AT && BT values: 5 3
enter AT && BT values: 4 1
enter AT && BT values: 2 6
Gantt_Chart is: idle p7 p6 p2 p1 p5 p3 p4
P       AT      BT      CT      TAT     WT
p1      8       2       13      5       3
p2      3       2       11      8       6
p3      6       5       21      15      10
p4      2       8       29      27      19
p5      5       3       16      11      8
p6      4       1       9       5       4
p7      2       6       8       6       0

average waiting time: 7.14
average Turn around time: 11.00
--------------------------------
Process exited after 17.7 seconds with return value 31
Press any key to continue . . .
```

3.PRIORITY

```
#include<stdio.h>
int find_min_at(int n,int arr[][7],int i,int visit[])
{
        int min=9999,flag,ct=0;
        int j;
        for(j=1;j<=n;j++)
        {
                if(arr[j][1]<min && visit[j]==-1)
                {
                        min=arr[j][i];
                        flag=j;
                }
        }
        return flag;
}
int find_min_priority(int n,int arr[][7],int i,int visit[],int ct)
{
        int min=9999,flag=-1;
        int j;
        for(j=1;j<=n;j++)
        {
                if((arr[j][i]<min  ||  (arr[j][i]==min  &&  arr[j][2]<arr[flag][2]))  &&  visit[j]==-1
&& arr[j][2]<=ct)
                {
                        min=arr[j][i];
                        flag=j;
                }
        }
        return flag;

}
int main()
{
        int n,i,ct=0,j;
        printf("enter the number of procesors: ");
        scanf("%d",&n);
        int arr[n][7],gantt_chart[100],visit[n];
        //P     PRIORITY AT     BT      CT      TAT     WT

        for(i=1;i<=n;i++)
        {
                arr[i][0]=i;
                visit[i]=-1;
        }
        printf("enter the values: \n");
        for(i=1;i<=n;i++)
        {
                printf("enter Priority && AT && BT values: ");
                scanf("%d %d %d",&arr[i][1],&arr[i][2],&arr[i][3]);
        }
```

```
int p=-1,l=0,k=0;
int min=999;
for(i=1;i<=n;i++)
{
        if(min>arr[i][2] || (min==arr[i][2] && arr[p][1]>arr[i][1]))
        {
                min=arr[i][2];
                p=i;
        }
}
ct=min;
if(min!=0)
{
        gantt_chart[l]=-1;
        l++;
        gantt_chart[l]=p;
        printf("%d\n",gantt_chart[l]);
        l++;
        visit[p]=1;
        arr[p][4]=ct+arr[p][3];
        ct=arr[p][4];
        k++;
}
else
{
        gantt_chart[l]=p;
        l++;
        k++;
        visit[p]=1;
        arr[p][4]=arr[p][3];
        ct=arr[p][4];
}
while(k!=n)
{
        p=find_min_priority(n,arr,1,visit,ct);
        if(p==-1)
        {
                gantt_chart[l]=-1;
                l++;
                p=find_min_at(n,arr,2,visit);
                ct=arr[p][2];
                p=find_min_priority(n,arr,1,visit,ct);
        }
        visit[p]=1;
        k++;
        gantt_chart[l]=p;
        l++;
        arr[p][4]=ct+arr[p][3];
        ct=arr[p][4];
}
printf("Gantt_Chart is: ");
```

```
        for(i=0;i<l;i++)
        {
                if(gantt_chart[i]==-1)
                {
                        printf("idle ");
                }
                else
                {
                        printf("p%d ",gantt_chart[i]);
                }
        }
        float sum_wt=0,sum_tat=0;
        for(i=1;i<=n;i++)
        {
                arr[i][5]=arr[i][4]-arr[i][2];
                arr[i][6]=arr[i][5]-arr[i][3];
                sum_wt+=arr[i][6];
                sum_tat+=arr[i][5];

        }
        printf("\n");
        printf("P\tPrior\tAT\tBT\tCT\tTAT\tWT\n");
        for(i=1;i<=n;i++)
        {       printf("p%d       ",i);
                for(j=1;j<7;j++)
                {
                        printf("%d\t",arr[i][j]);
                }
                printf("\n");
        }

        printf("\naverage waiting time: %f",sum_wt/n);
        printf("\naverage Turn around time: %f",sum_tat/n);
}
```

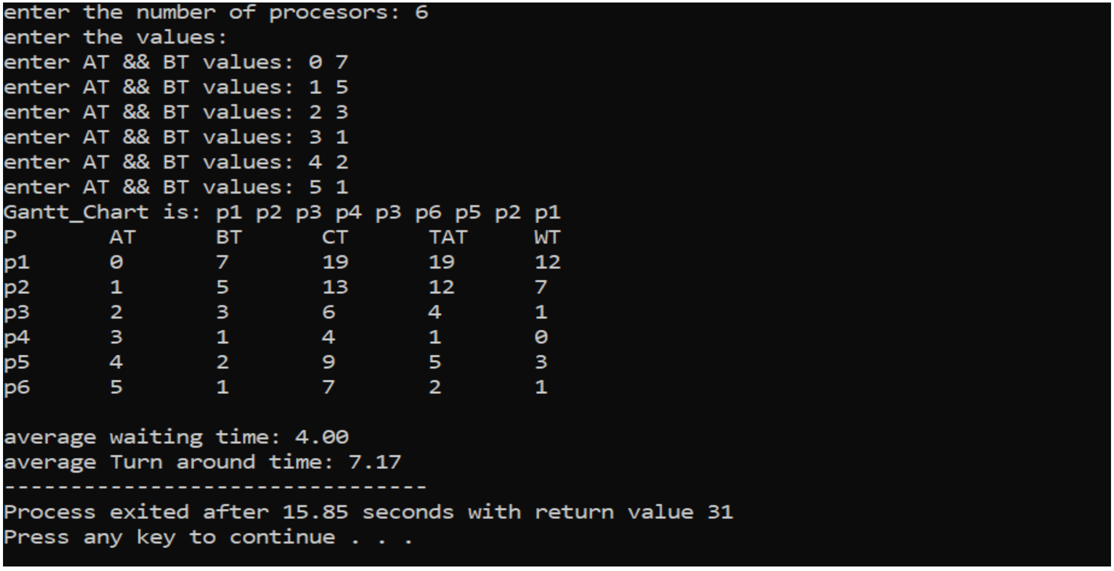


```
enter the number of procesors: 7
enter the values:
enter Priority && AT && BT values: 2 0 3
enter Priority && AT && BT values: 6 2 5
enter Priority && AT && BT values: 3 1 4
enter Priority && AT && BT values: 5 4 2
enter Priority && AT && BT values: 7 6 9
enter Priority && AT && BT values: 4 5 4
enter Priority && AT && BT values: 10 7 10
Gantt_Chart is: p1 p3 p6 p4 p2 p5 p7
P       Prior   AT      BT      CT      TAT     WT
p1      2       0       3       3       3       0
p2      6       2       5       18      16      11
p3      3       1       4       7       6       2
p4      5       4       2       13      9       7
p5      7       6       9       27      21      12
p6      4       5       4       11      6       2
p7      10      7       10      37      30      20

average waiting time: 7.714286
average Turn around time: 13.000000
--------------------------------
Process exited after 48.81 seconds with return value 36
Press any key to continue . . .
```

3.SRTF(SHORTEST REMAIN TIME FIRST)

```
#include<stdio.h>
int find_min_at(int n,int arr[][6],int i,int visit[])
{
        int min=9999,flag;
        int j;
        for(j=1;j<=n;j++)
        {
                if(arr[j][1]<min && visit[j]==-1)
                {
                        min=arr[j][i];
                        flag=j;
                }
        }
        return flag;
}
int find_min(int n,int arr[][6],int i,int visit[],int ct)
{
        int min=9999,flag=-1;
        int j;
        for(j=1;j<=n;j++)
        {
                if((arr[j][i]<min || (arr[j][i]==min && arr[j][1]<=arr[flag][1])) && visit[j]==-1
&& arr[j][1]<=ct)
                {
                        min=arr[j][i];
                        flag=j;
                }
        }
        if(flag==-1)
        {
                return -1;
        }
        else
        {
                return flag;
        }

}
int main()
{
        int n,i,ct=0,j;
        printf("enter the number of procesors: ");
        scanf("%d",&n);
        int arr[n][6],gantt_chart[100],visit[n];
        int bt[n];
        //P     AT      BT      CT      TAT     WT

        for(i=1;i<=n;i++)
        {
                arr[i][0]=i;
                visit[i]=-1;
        }
        printf("enter the values: \n");
        for(i=1;i<=n;i++)
        {
                printf("enter AT && BT values: ");
                scanf("%d %d",&arr[i][1],&arr[i][2]);
                arr[i][3]=0;
                bt[i]=arr[i][2];
        }
        int p=-1,l=0,k=0;
        int min=999;
        for(i=1;i<=n;i++)
        {
                if(min>arr[i][1] || (min==arr[i][1] && arr[p][2]>arr[i][2]))
                {
                        min=arr[i][1];
                        p=i;
                }
        }
        ct=min;
        if(min!=0)
        {
                gantt_chart[l]=-1;
                l++;
                gantt_chart[l]=p;
                l++;
                arr[p][2]=arr[p][2]-1;
                arr[p][3]++;
                ct=arr[p][3];
        }
        else
        {
                gantt_chart[l]=p;
                l++;
                arr[p][2]=arr[p][2]-1;
                arr[p][3]++;
                ct=arr[p][3];
        }
        if(arr[p][2]==0)
        {
                visit[p]=1;
                k++;
        }
        while(k!=n)
        {
                p=find_min(n,arr,2,visit,ct);
                if(p==-1)
                {
                        gantt_chart[l]=-1;
                        l++;
                        p=find_min_at(n,arr,1,visit);
                        ct=arr[p][1];
                        p=find_min(n,arr,2,visit,ct);
                }
                arr[p][2]--;
                if(gantt_chart[l-1]==p)
                {
                        arr[p][3]=ct+1;
                        ct++;
                }
                else
                {
                        gantt_chart[l]=p;
                        l++;
                        arr[p][3]=ct+1;
                        ct++;
                }
                if(arr[p][2]==0)
                {
                        visit[p]=1;
                        k++;
                }
        }
        float sum_wt=0,sum_tat=0;
        for(i=1;i<=n;i++)
        {
                arr[i][4]=arr[i][3]-arr[i][1];
                arr[i][5]=arr[i][4]-bt[i];
                sum_wt+=arr[i][5];
                sum_tat+=arr[i][4];
        }
        printf("Gantt_Chart is: ");
        for(i=0;i<l;i++)
        {
                if(gantt_chart[i]==-1)
                {
                        printf("idle ");
                }
                else
                        printf("p%d ",gantt_chart[i]);
        }
        printf("\n");
        printf("P\tAT\tBT\tCT\tTAT\tWT\n");
        for(i=1;i<=n;i++)
        {
                printf("p%d       ",i);
                arr[i][2]=bt[i];
                for(j=1;j<6;j++)
                {
                        printf("%d\t",arr[i][j]);
                }
```

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

        printf("\naverage waiting time: %.2f",sum_wt/n);
        printf("\naverage Turn around time: %.2f",sum_tat/n);
}
```



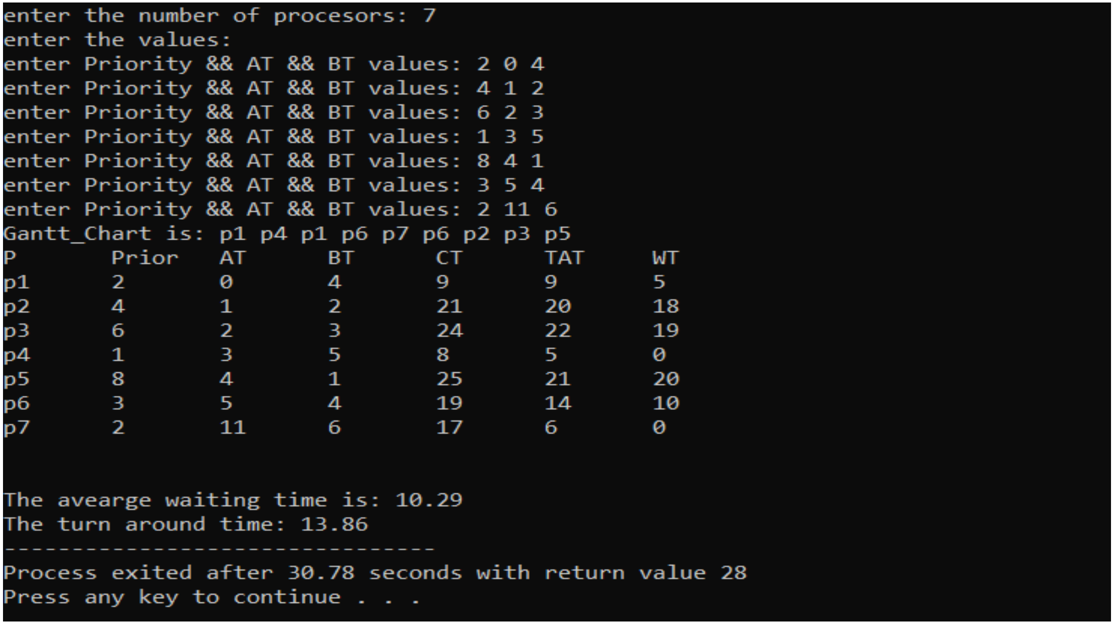
```
enter the number of procesors: 6
enter the values:
enter AT && BT values: 0 7
enter AT && BT values: 1 5
enter AT && BT values: 2 3
enter AT && BT values: 3 1
enter AT && BT values: 4 2
enter AT && BT values: 5 1
Gantt_Chart is: p1 p2 p3 p4 p3 p6 p5 p2 p1
P       AT      BT      CT      TAT     WT
p1      0       7       19      19      12
p2      1       5       13      12      7
p3      2       3       6       4       1
p4      3       1       4       1       0
p5      4       2       9       5       3
p6      5       1       7       2       1

average waiting time: 4.00
average Turn around time: 7.17
--------------------------------
Process exited after 15.85 seconds with return value 31
Press any key to continue . . .
```

5. PREEMTIVE PREORITY:

```
#include<stdio.h>
int find_min_at(int n,int arr[][7],int i,int visit[])
{
        int min=9999,flag,ct=0;
        int j;
        for(j=1;j<=n;j++)
        {
                if((min>arr[i][2] || (min==arr[i][2] && (arr[flag][1]>arr[i][1]
||(arr[flag][1]==arr[i][1] && arr[flag][3]>arr[i][3])))) && visit[j]==-1)
                {
                        min=arr[j][i];
                        flag=j;
                }
        }
        return flag;
}
int find_min_priority(int n,int arr[][7],int i,int visit[],int ct)
{
        int min=9999,flag=-1;
        int j;
        for(j=1;j<=n;j++)
        {
                if((arr[j][i]<min || (arr[j][i]==min && (arr[j][3]<arr[flag][3] ||
(arr[j][3]==arr[flag][3] && arr[j][2]<arr[flag][2])))) && visit[j]==-1 && arr[j][2]<=ct)
                {
                        min=arr[j][i];
```

```
            flag=j;
        }
    }
    if(flag==-1)
    {
        return -1;
    }
    else
    {
        return flag;
    }

}
int main()
{
    int n,i,ct=0,j;
    printf("enter the number of procesors: ");
    scanf("%d",&n);
    int arr[n][7],gantt_chart[100],visit[n];
    int bt[n];
    //P     PRIORITY AT     BT      CT      TAT     WT

    for(i=1;i<=n;i++)
    {
        arr[i][0]=i;
        visit[i]=-1;
    }
    printf("enter the values: \n");
    for(i=1;i<=n;i++)
    {
        printf("enter Priority && AT && BT values: ");
        scanf("%d %d %d",&arr[i][1],&arr[i][2],&arr[i][3]);
        arr[i][4]=0;
        bt[i]=arr[i][3];
    }
    int p=-1,l=0,k=0;
    int min=999;
    for(i=1;i<=n;i++)
    {
        if(min>arr[i][2] || (min==arr[i][2] && (arr[p][1]>arr[i][1]
||(arr[p][1]==arr[i][1] && arr[p][3]>arr[i][3]))))
        {
            min=arr[i][2];
            p=i;
        }
    }
    ct=0;
    if(min!=0)
    {
        ct=min;
        gantt_chart[l]=-1;
        l++;
        gantt_chart[l]=p;
        l++;
        ct++;
        arr[p][3]--;
        arr[p][4]=ct;
    }
    else
    {
        gantt_chart[l]=p;
        l++;
        arr[p][3]--;
        ct++;
        arr[p][4]=ct;
    }
    if(arr[p][3]==0)
    {
        k++;
        visit[p]=1;
    }
    while(k!=n)
    {
        p=find_min_priority(n,arr,1,visit,ct);
        if(p==-1)
        {
            gantt_chart[l]=-1;
            l++;
            p=find_min_at(n,arr,2,visit);
            ct=arr[p][2];
            p=find_min_priority(n,arr,1,visit,ct);
        }
        arr[p][3]--;
        if(gantt_chart[l-1]==p)
        {
            ct++;
            arr[p][4]=ct;
        }
        else
        {
            gantt_chart[l]=p;
            l++;
            ct++;
            arr[p][4]=ct;

        }
        if(arr[p][3]==0)
        {
            visit[p]=1;
            k++;
        }
    }
```

```
	float sum_wt=0,sum_tat=0;
	for(i=1;i<=n;i++)
	{
		arr[i][5]=arr[i][4]-arr[i][2];
		arr[i][6]=arr[i][5]-bt[i];
		sum_wt+=arr[i][6];
		sum_tat+=arr[i][5];
	}
	printf("Gantt_Chart is: ");
	for(i=0;i<l;i++)
	{
		if(gantt_chart[i]==-1)
		{
			printf("idle ");
		}
		else
			printf("p%d ",gantt_chart[i]);
	}
	printf("\n");
	printf("P\tPrior\tAT\tBT\tCT\tTAT\tWT\n");
	for(i=1;i<=n;i++)
	{
		printf("p%d       ",i);
		arr[i][3]=bt[i];
		for(j=1;j<7;j++)
		{
			printf("%d\t",arr[i][j]);
		}
		printf("\n");
	}

	printf("\n\nThe avearge waiting time is: %.2f",sum_wt/n);
	printf("\nThe turn around time: %.2f",sum_tat/n);
}
```



```
enter the number of procesors: 7
enter the values:
enter Priority && AT && BT values: 2 0 4
enter Priority && AT && BT values: 4 1 2
enter Priority && AT && BT values: 6 2 3
enter Priority && AT && BT values: 1 3 5
enter Priority && AT && BT values: 8 4 1
enter Priority && AT && BT values: 3 5 4
enter Priority && AT && BT values: 2 11 6
Gantt_Chart is: p1 p4 p1 p6 p7 p6 p2 p3 p5
P       Prior   AT      BT      CT      TAT     WT
p1      2       0       4       9       9       5
p2      4       1       2       21      20      18
p3      6       2       3       24      22      19
p4      1       3       5       8       5       0
p5      8       4       1       25      21      20
p6      3       5       4       19      14      10
p7      2       11      6       17      6       0


The avearge waiting time is: 10.29
The turn around time: 13.86
--------------------------------
Process exited after 30.78 seconds with return value 28
Press any key to continue . . .
```

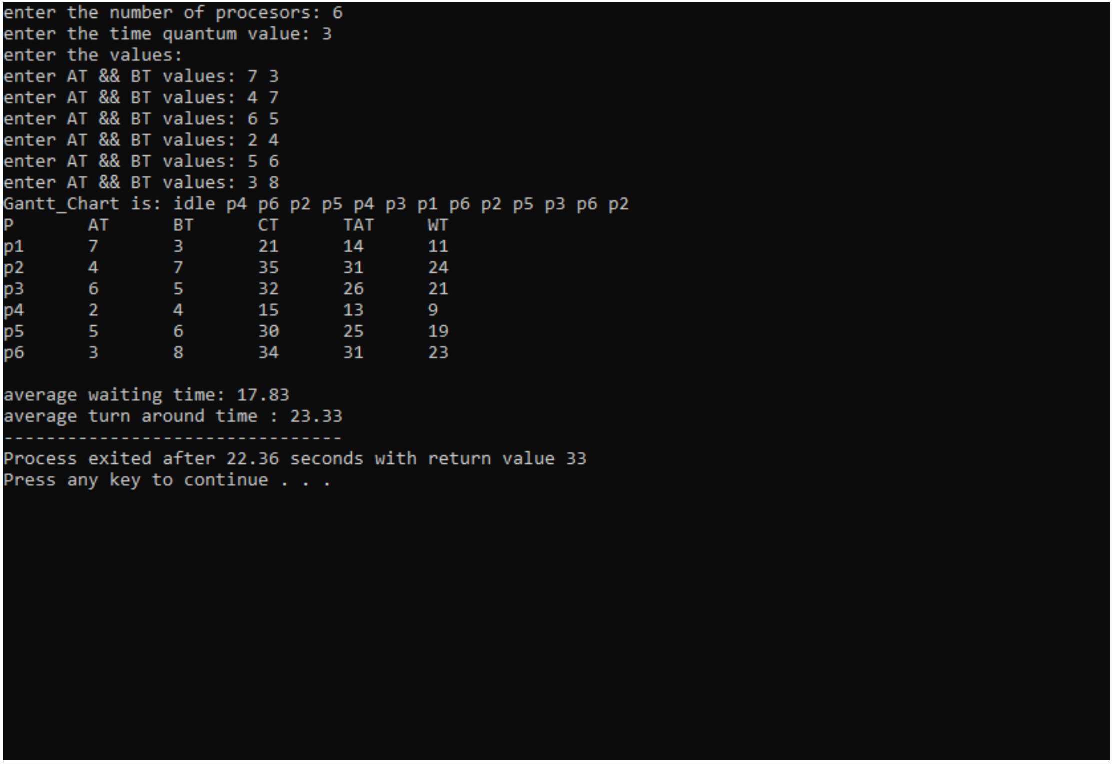
6.ROUND ROBIN(RR)

```
 #include<stdio.h>
int queue[100];
int st=-1,end=-1;
int e=0;
int find_min(int n,int arr[][6],int i,int vis[])
{
        int min=9999,flag;
        int j;
        for(j=1;j<=n;j++)
        {
                if(arr[j][1]<min && vis[j]==-1)
                {
                        min=arr[j][1];
                        flag=j;
                }
        }
        return flag;
}
int find_min1(int n,int arr[][6],int visit[],int ct)
{
        int i,min=9999,flag=-1;
        for(i=1;i<=n;i++)
        {
                if(arr[i][1]<min && arr[i][1]<=ct && visit[i]==-1)
                {
                        min=arr[i][1];
                        flag=i;
                }
        }
        return flag;
}
int find_process(int n,int arr[][6],int visit[],int ct)
{
        int p=find_min1(n,arr,visit,ct);
        while(p!=-1)
        {
                enque(p);
                visit[p]=1;
                e++;
                p=find_min1(n,arr,visit,ct);
        }
}
int enque(int p)
{
        if(st==-1)
        {
                st=0;
                end=0;
                queue[end]=p;
        }
        else{
                end++;
                queue[end]=p;
        }
}
int deque()
{
        if(st==-1)
        {
                return -1;
        }
        if(st==end)
        {
                int r=queue[st];
                st=-1;
                end=-1;
                return r;
        }
        else{
                int r=queue[st];
                st++;
                return  r;
        }
}
int main()
{
        int n,i,ct=0,j,TQ;
        printf("enter the number of procesors: ");
        scanf("%d",&n);
        int arr[n][6],gantt_chart[100],visit[n],vis[n];
        int bt[n];
        printf("enter the time quantum value: ");
        scanf("%d",&TQ);
        //P       AT      BT      CT      TAT     WT

        for(i=1;i<=n;i++)
        {
                arr[i][0]=i;
                visit[i]=-1;
                vis[i]=-1;
        }
        printf("enter the values: \n");
        for(i=1;i<=n;i++)
        {
                printf("enter AT && BT values: ");
                scanf("%d %d",&arr[i][1],&arr[i][2]);
                bt[i]=arr[i][2];
        }
        int p=-1,l=0,k=0;
        int min=999;
        for(i=1;i<=n;i++)
```

```
	{
		if(min>arr[i][1])
		{
			min=arr[i][1];
		}
	}
	ct=0;
	if(min!=0)
	{
		ct=min;
		gantt_chart[l]=-1;
		l++;
	}
	find_process(n,arr,visit,ct);
	p=deque();
	gantt_chart[l]=p;
	l++;
	if(arr[p][2]>TQ)
	{
		arr[p][2]=(arr[p][2]-TQ);
		ct=ct+TQ;
	}
	else{
		ct=ct+arr[p][2];
		arr[p][3]=ct;
		arr[p][2]=0;
		vis[p]=1;
		k++;
	}
	while(k!=n)
	{
		if(e!=n)
		{
			find_process(n,arr,visit,ct);
			if(arr[p][2]!=0)
			{
				enque(p);
			}
		}
		p=deque();
		if(p==-1)
		{
			gantt_chart[l]=-1;
			l++;
			p=find_min(n,arr,1,vis);
			ct=arr[p][1];
			find_process(n,arr,visit,ct);
			p=deque();
		}
		if(gantt_chart[l-1]!=p)
		{
```

```
                gantt_chart[l]=p;
                l++;
            }
            if(arr[p][2]>TQ)
            {
                arr[p][2]=arr[p][2]-TQ;
                ct=ct+TQ;
                if(e==n)
                {
                    enque(p);
                }
            }
            else{
                ct=ct+arr[p][2];
                arr[p][3]=ct;
                arr[p][2]=0;
                vis[p]=1;
                k++;
            }
        }
        float sum_wt=0,sum_tat=0;
        for(i=1;i<=n;i++)
        {
            arr[i][4]=arr[i][3]-arr[i][1];
            arr[i][5]=arr[i][4]-bt[i];
            sum_wt+=arr[i][5];
            sum_tat+=arr[i][4];
        }
        printf("Gantt_Chart is: ");
        for(i=0;i<l;i++)
        {
            if(gantt_chart[i]==-1)
            {
                printf("idle ");
            }
            else
                printf("p%d ",gantt_chart[i]);
        }
        printf("\n");
        printf("P\tAT\tBT\tCT\tTAT\tWT\n");
        for(i=1;i<=n;i++)
        {
            printf("p%d       ",i);
            arr[i][2]=bt[i];
            for(j=1;j<6;j++)
            {
                printf("%d\t",arr[i][j]);
            }
            printf("\n");
        }
```

```
	printf("\naverage waiting time: %.2f",sum_wt/n);
	printf("\naverage turn around time : %.2f",sum_tat/n);

}
```



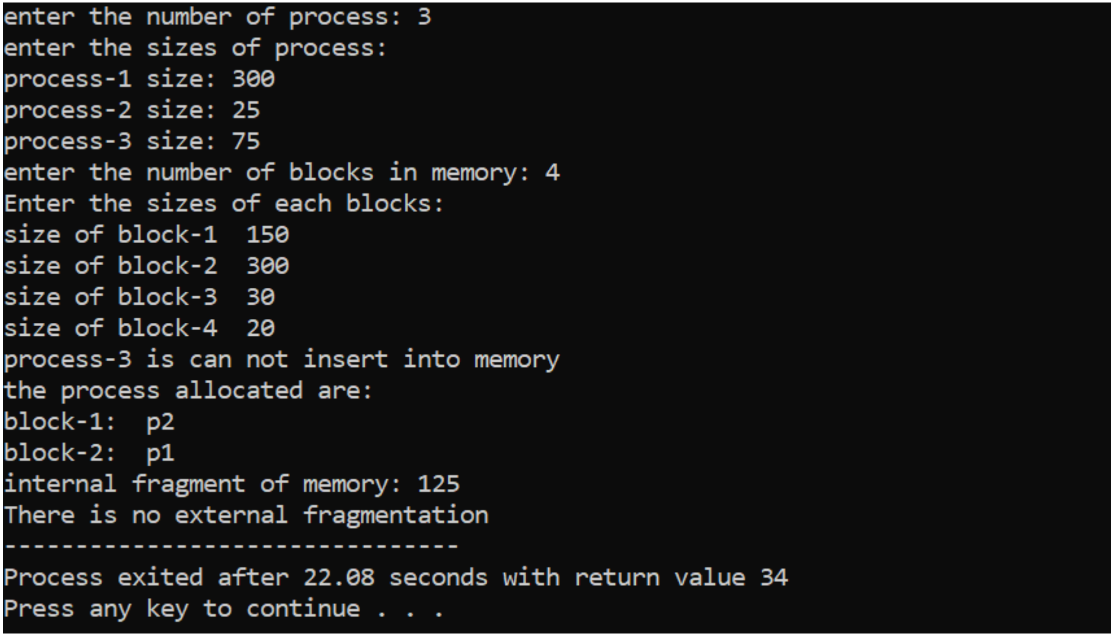
```
enter the number of procesors: 6
enter the time quantum value: 3
enter the values:
enter AT && BT values: 7 3
enter AT && BT values: 4 7
enter AT && BT values: 6 5
enter AT && BT values: 2 4
enter AT && BT values: 5 6
enter AT && BT values: 3 8
Gantt_Chart is: idle p4 p6 p2 p5 p4 p3 p1 p6 p2 p5 p3 p6 p2
P       AT      BT      CT      TAT     WT
p1      7       3       21      14      11
p2      4       7       35      31      24
p3      6       5       32      26      21
p4      2       4       15      13      9
p5      5       6       30      25      19
p6      3       8       34      31      23

average waiting time: 17.83
average turn around time : 23.33
--------------------------------
Process exited after 22.36 seconds with return value 33
Press any key to continue . . .
```

Memory Allocations

1. Fixed Partition First Fit:

```
#include<stdio.h>
int main()
{
	int n,i,j;
	printf("enter the number of process: ");
	scanf("%d",&n);
	int m,size[n],a[n];
	printf("enter the sizes of process: \n");
	for(i=1;i<=n;i++)
	{
		printf("process-%d size: ",i);
		scanf("%d",&size[i]);
	}
	printf("enter the number of blocks in memory: ");
	scanf("%d",&m);
	int block[m],visit[m];
	printf("Enter the sizes of each blocks: \n");
	for(i=1;i<=m;i++)
	{
		printf("size of block-%d  ",i);
		scanf("%d",&block[i]);
```

```
                visit[i]=-1;
        }
        int k=1,count=0;
        while(k<=n){
                for(i=1;i<=m;i++)
                {
                        if(block[i]>=size[k] && visit[i]==-1)
                        {
                                visit[i]=k;
                                block[i]=block[i]-size[k];
                                k++;
                                break;
                        }
                }
                if(i>m)
                {
                        a[count]=k;
                        count++;
                        printf("process-%d is can not insert into memory\n",k);
                        k++;
                }
        }
        printf("the process allocated are: \n");
        for(i=1;i<=m;i++)
        {
                if(visit[i]!=-1)
                {
                        printf("block-%d:  p%d\n",i,visit[i]);}
        }
        int sum=0;
        printf("internal fragment of memory: ");
        for(i=1;i<=m;i++)
        {
                if(visit[i]!=-1)
                {
                        sum=sum+block[i];
                }
        }
        printf("%d\n",sum);
        if(count!=0)
        {
                sum=0;
                for(i=1;i<=m;i++)
                {
                        if(visit[i]==-1){
                                sum=sum+block[i];
                        }
                }
                for(i=0;i<count;i++)
                {
                        if(sum>=size[a[i]])
                        {
                                break;
                        }
                }
                if(i<count)
                {
                        printf("external segment: %d",sum);
                }
                else{
                        printf("There is no external fragmentation");
                }
        }

}
```

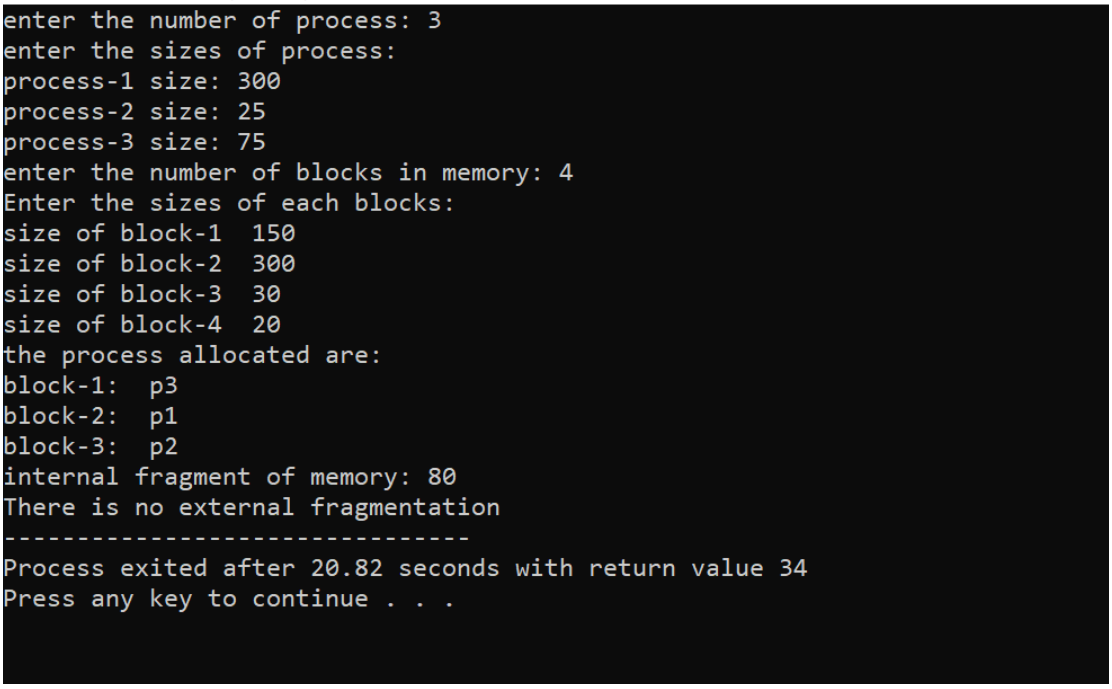


```
enter the number of process: 3
enter the sizes of process:
process-1 size: 300
process-2 size: 25
process-3 size: 75
enter the number of blocks in memory: 4
Enter the sizes of each blocks:
size of block-1  150
size of block-2  300
size of block-3  30
size of block-4  20
process-3 is can not insert into memory
the process allocated are:
block-1:  p2
block-2:  p1
internal fragment of memory: 125
There is no external fragmentation
--------------------------------
Process exited after 22.08 seconds with return value 34
Press any key to continue . . .
```

2. Fixed Partition Best Fit:

```
#include<stdio.h>
int main()
{
        int n,i,j;
        printf("enter the number of process: ");
        scanf("%d",&n);
        int m,size[n];
        printf("enter the sizes of process: \n");
        for(i=1;i<=n;i++)
        {
                printf("process-%d size: ",i);
                scanf("%d",&size[i]);
        }
        printf("enter the number of blocks in memory: ");
        scanf("%d",&m);
        int block[m],visit[m];
```

```
    printf("Enter the sizes of each blocks: \n");
    for(i=1;i<=m;i++)
    {
        printf("size of block-%d  ",i);
        scanf("%d",&block[i]);
        visit[i]=-1;
    }
    int a[n];
    int k=1,count=0;
    while(k<=n){
        int min=999,flag=0;
        for(i=1;i<=m;i++)
        {
            if(block[i]<min && block[i]>=size[k] && visit[i]==-1)
            {
                min=block[i];
                flag=i;
            }
        }
        if(flag!=0)
        {
            visit[flag]=k;
            block[flag]=block[flag]-size[k];
            k++;
        }
        else
        {
            a[count]=k;
            count++;
            printf("process-%d is can not insert into memory\n",k);
            k++;
        }
    }
    printf("the process allocated are: \n");
    for(i=1;i<=m;i++)
    {
        if(visit[i]!=-1)
        {
            printf("block-%d:  p%d\n",i,visit[i]);
        }
    }
    int sum=0;
    printf("internal fragment of memory: ");
    for(i=1;i<=m;i++)
    {
        if(visit[i]!=-1)
        {
            sum=sum+block[i];
        }
    }
    printf("%d\n",sum);
    if(count!=0)
    {
        sum=0;
        for(i=1;i<=m;i++)
        {
            if(visit[i]==-1){
                sum=sum+block[i];
            }
        }
        for(i=0;i<count;i++)
        {
            if(sum>=size[a[i]])
            {
                break;
            }
        }
        if(i<count)
        {
            printf("external fragment: %d",sum);
        }
        else
        {
            printf("There is no external fragmentation");
        }
    }
    else
    {
        printf("There is no external fragmentation");
    }
}
```



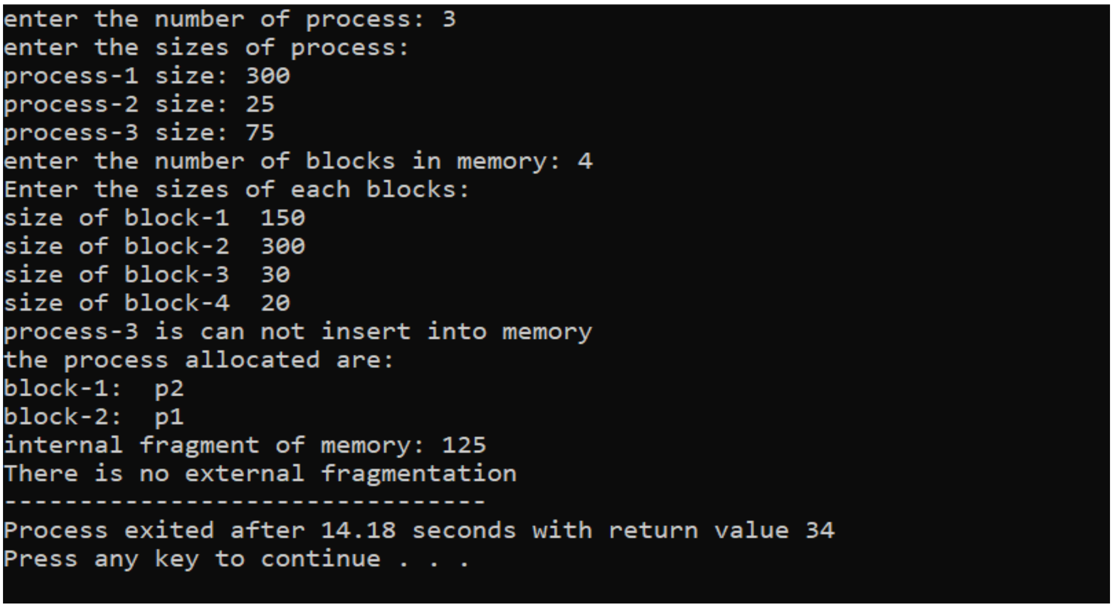
```
enter the number of process: 3
enter the sizes of process:
process-1 size: 300
process-2 size: 25
process-3 size: 75
enter the number of blocks in memory: 4
Enter the sizes of each blocks:
size of block-1  150
size of block-2  300
size of block-3  30
size of block-4  20
the process allocated are:
block-1:  p3
block-2:  p1
block-3:  p2
internal fragment of memory: 80
There is no external fragmentation
--------------------------------
Process exited after 20.82 seconds with return value 34
Press any key to continue . . .
```

3. Fixed Partition Worst Fit:

```
#include<stdio.h>
int main()
{
    int n,i,j;
    printf("enter the number of process: ");
    scanf("%d",&n);
    int m,size[n];
    printf("enter the sizes of process: \n");
    for(i=1;i<=n;i++)
    {
        printf("process-%d size: ",i);
        scanf("%d",&size[i]);
    }
    printf("enter the number of blocks in memory: ");
    scanf("%d",&m);
    int block[m],visit[m];
    printf("Enter the sizes of each blocks: \n");
    for(i=1;i<=m;i++)
    {
        printf("size of block-%d  ",i);
        scanf("%d",&block[i]);
        visit[i]=-1;
    }
    int a[n];
    int k=1,count=0;
    while(k<=n){
        int min=0,flag=0;
        for(i=1;i<=m;i++)
        {
            if(block[i]>min && block[i]>=size[k] && visit[i]==-1)
            {
                min=block[i];
                flag=i;
            }
        }
        if(flag!=0)
        {
            visit[flag]=k;
            block[flag]=block[flag]-size[k];
            k++;
        }
        else
        {
            a[count]=k;
            count++;
            printf("process-%d is can not insert into memory\n",k);
            k++;
        }
    }
```

```
        printf("the process allocated are: \n");
        for(i=1;i<=m;i++)
        {
                if(visit[i]!=-1)
                {
                        printf("block-%d:  p%d\n",i,visit[i]);
                }
        }
        int sum=0;
        printf("internal fragment of memory: ");
        for(i=1;i<=m;i++)
        {
                if(visit[i]!=-1)
                {
                        sum=sum+block[i];
                }
        }
        printf("%d\n",sum);
        if(count!=0)
        {
                sum=0;
                for(i=1;i<=m;i++)
                {
                        if(visit[i]==-1){
                                sum=sum+block[i];
                        }
                }
                for(i=0;i<count;i++)
                {
                        if(sum>=size[a[i]])
                        {
                                break;
                        }
                }
                if(i<count)
                {
                        printf("external fragment: %d",sum);
                }
                else
                {
                        printf("There is no external fragmentation");
                }
        }
        else
        {
                printf("There is no external fragmentation");
        }
```

}



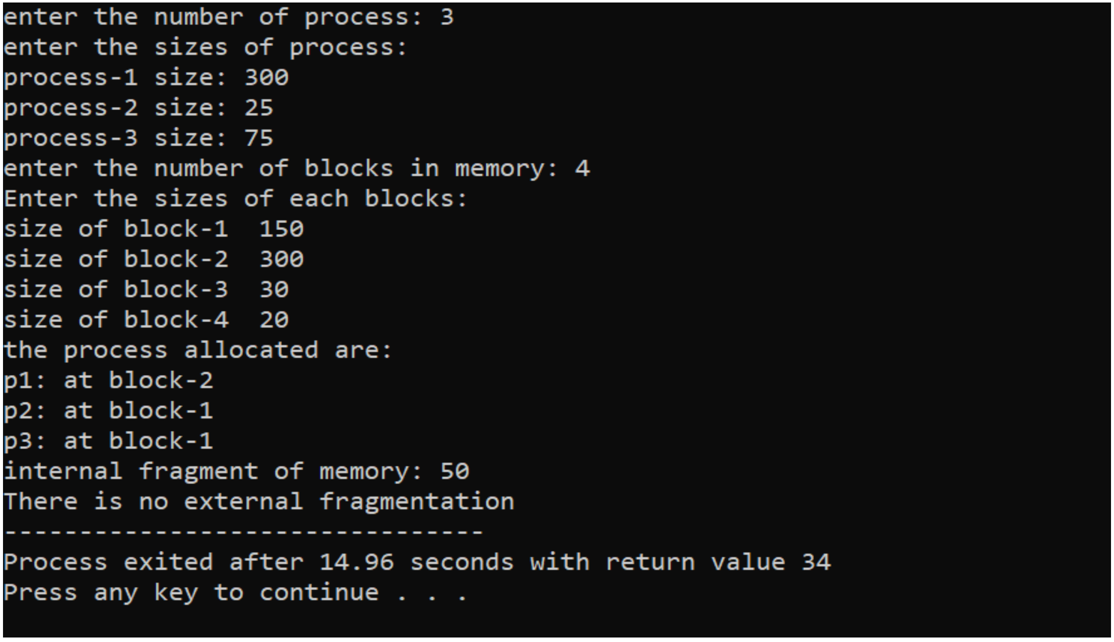
```
enter the number of process: 3
enter the sizes of process:
process-1 size: 300
process-2 size: 25
process-3 size: 75
enter the number of blocks in memory: 4
Enter the sizes of each blocks:
size of block-1  150
size of block-2  300
size of block-3  30
size of block-4  20
process-3 is can not insert into memory
the process allocated are:
block-1:  p2
block-2:  p1
internal fragment of memory: 125
There is no external fragmentation
--------------------------------
Process exited after 14.18 seconds with return value 34
Press any key to continue . . .
```

4. Variable Partition First Fit:

```
#include<stdio.h>
int main()
{
        int n,i,j;
        printf("enter the number of process: ");
        scanf("%d",&n);
        int m,size[n],a[n],store[n];
        printf("enter the sizes of process: \n");
        for(i=1;i<=n;i++)
        {
                printf("process-%d size: ",i);
                scanf("%d",&size[i]);
                store[i]=-1;
        }
        printf("enter the number of blocks in memory: ");
        scanf("%d",&m);
        int block[m],visit[m];
        printf("Enter the sizes of each blocks: \n");
        for(i=1;i<=m;i++)
        {
                printf("size of block-%d  ",i);
                scanf("%d",&block[i]);
                visit[i]=-1;
        }
        int k=1,count=0;
        while(k<=n){
                for(i=1;i<=m;i++)
                {
                        if(block[i]>=size[k])
                        {
```

```
                        store[k]=i;
                        block[i]=block[i]-size[k];
                        k++;
                        visit[i]=1;
                        break;
                    }
                }
                if(i>m)
                {
                    a[count]=k;
                    count++;
                    printf("process-%d is can not insert into memory\n",k);
                    k++;
                }
            }
            printf("the process allocated are: \n");
            for(i=1;i<k;i++)
            {
                if(store[i]!=-1)
                {
                    printf("p%d: at block-%d\n",i,store[i]);
                }
            }
            int sum=0;
            printf("internal fragment of memory: ");
            for(i=1;i<=m;i++)
            {
                if(visit[i]!=-1)
                {
                    sum=sum+block[i];
                }
            }
            printf("%d\n",sum);
            if(count!=0)
            {
                sum=0;
                for(i=1;i<=m;i++)
                {
                    if(visit[i]==-1){
                        sum=sum+block[i];
                    }
                }
                for(i=0;i<count;i++)
                {
                    if(sum>=size[a[i]])
                    {
                        break;
                    }
                }
                if(i<count)
                {
```

```
                printf("external segment: %d",sum);
            }
            else
            {
                printf("There is no external fragmentation");
            }
        }
        else
        {
            printf("There is no external fragmentation");
        }
}
```



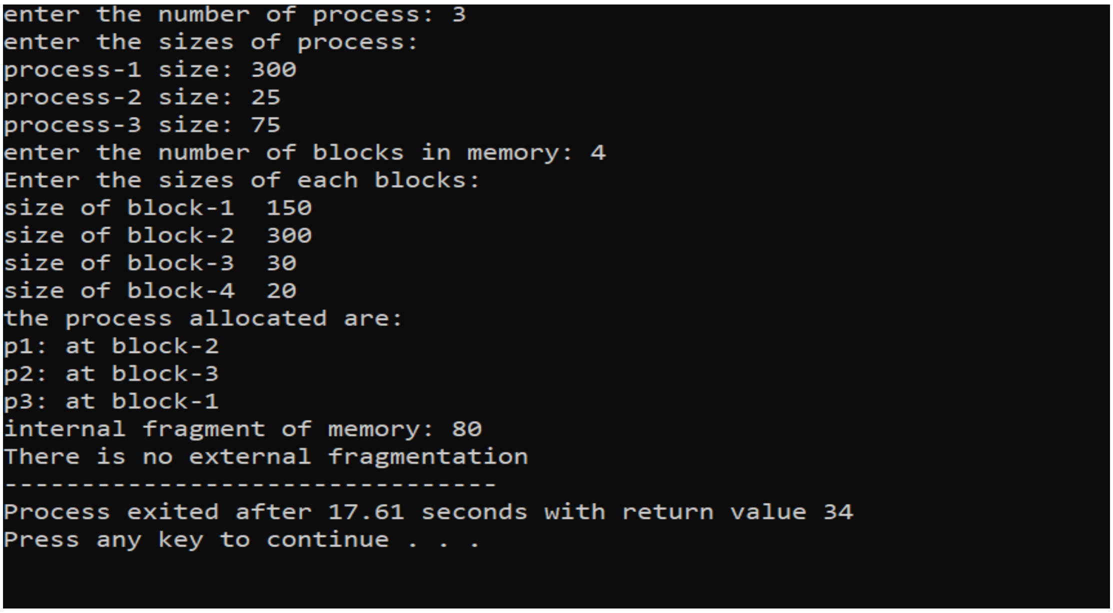
```
enter the number of process: 3
enter the sizes of process:
process-1 size: 300
process-2 size: 25
process-3 size: 75
enter the number of blocks in memory: 4
Enter the sizes of each blocks:
size of block-1  150
size of block-2  300
size of block-3  30
size of block-4  20
the process allocated are:
p1: at block-2
p2: at block-1
p3: at block-1
internal fragment of memory: 50
There is no external fragmentation
--------------------------------
Process exited after 14.96 seconds with return value 34
Press any key to continue . . .
```

5. Variable Partition Best Fit:

```
#include<stdio.h>
int main()
{
        int n,i,j;
        printf("enter the number of process: ");
        scanf("%d",&n);
        int m,size[n],a[n],store[n];
        printf("enter the sizes of process: \n");
        for(i=1;i<=n;i++)
        {
            printf("process-%d size: ",i);
            scanf("%d",&size[i]);
            store[i]=-1;
        }
        printf("enter the number of blocks in memory: ");
        scanf("%d",&m);
        int block[m],visit[m];
        printf("Enter the sizes of each blocks: \n");
```

```
for(i=1;i<=m;i++)
{
    printf("size of block-%d  ",i);
    scanf("%d",&block[i]);
    visit[i]=-1;
}
int k=1,count=0;
while(k<=n){
    int min=999,flag=0;
    for(i=1;i<=m;i++)
    {
        if(block[i]<min && block[i]>=size[k])
        {
            min=block[i];
            flag=i;
        }
    }
    if(flag!=0)
    {
        store[k]=flag;
        visit[flag]=1;
        block[flag]=block[flag]-size[k];
        k++;
    }
    else
    {
        a[count]=k;
        count++;
        printf("process-%d is can not insert into memory\n",k);
        k++;
    }
}
printf("the process allocated are: \n");
for(i=1;i<k;i++)
{
    if(store[i]!=-1)
    {
        printf("p%d: at block-%d\n",i,store[i]);
    }
}
int sum=0;
printf("internal fragment of memory: ");
for(i=1;i<=m;i++)
{
    if(visit[i]!=-1)
    {
        sum=sum+block[i];
    }
}
printf("%d\n",sum);
if(count!=0)
```

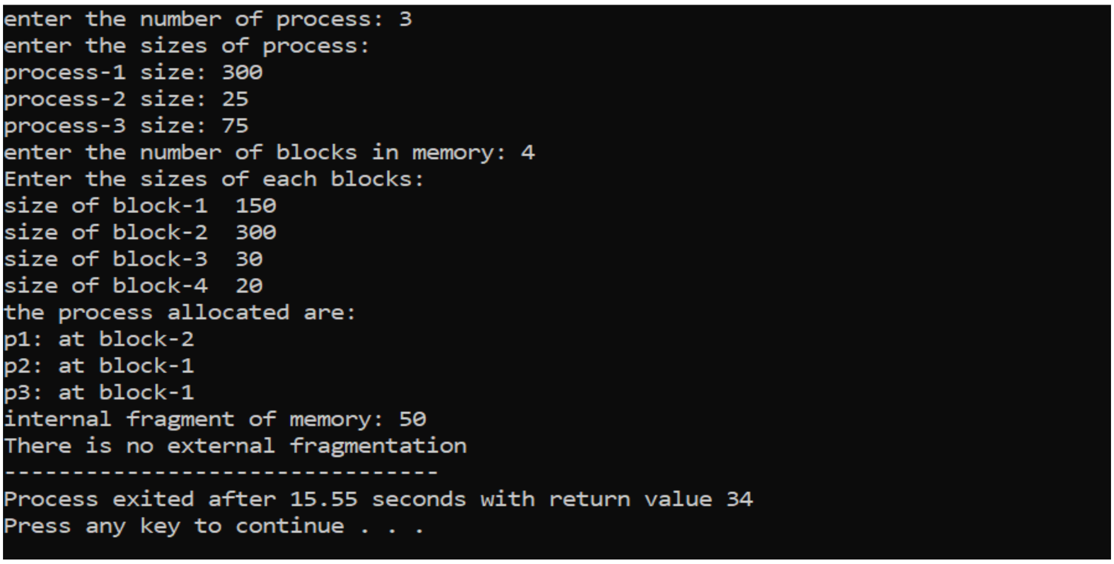
```
    {
        sum=0;
        for(i=1;i<=m;i++)
        {
            if(visit[i]==-1){
                sum=sum+block[i];
            }
        }
        for(i=0;i<count;i++)
        {
            if(sum>=size[a[i]])
            {
                break;
            }
        }
        if(i<count)
        {
            printf("external segment: %d",sum);
        }
        else
        {
            printf("There is no external fragmentation");
        }
    }
    else
    {
        printf("There is no external fragmentation");
    }

}
```



```
enter the number of process: 3
enter the sizes of process:
process-1 size: 300
process-2 size: 25
process-3 size: 75
enter the number of blocks in memory: 4
Enter the sizes of each blocks:
size of block-1  150
size of block-2  300
size of block-3  30
size of block-4  20
the process allocated are:
p1: at block-2
p2: at block-3
p3: at block-1
internal fragment of memory: 80
There is no external fragmentation
--------------------------------
Process exited after 17.61 seconds with return value 34
Press any key to continue . . .
```

6. Variable Partition Worst Fit:

```
#include<stdio.h>
int main()
{
        int n,i,j;
        printf("enter the number of process: ");
        scanf("%d",&n);
        int m,size[n],a[n],store[n];
        printf("enter the sizes of process: \n");
        for(i=1;i<=n;i++)
        {
                printf("process-%d size: ",i);
                scanf("%d",&size[i]);
                store[i]=-1;
        }
        printf("enter the number of blocks in memory: ");
        scanf("%d",&m);
        int block[m],visit[m];
        printf("Enter the sizes of each blocks: \n");
        for(i=1;i<=m;i++)
        {
                printf("size of block-%d  ",i);
                scanf("%d",&block[i]);
                visit[i]=-1;
        }
        int k=1,count=0;
        while(k<=n){
                int min=0,flag=0;
                for(i=1;i<=m;i++)
                {
                        if(block[i]>min && block[i]>=size[k])
                        {
                                min=block[i];
                                flag=i;
                        }
                }
                if(flag!=0)
                {
                        store[k]=flag;
                        visit[flag]=1;
                        block[flag]=block[flag]-size[k];
                        k++;
                }
                else
                {
                        a[count]=k;
                        count++;
                        printf("process-%d is can not insert into memory\n",k);
                        k++;
                }
        }
        printf("the process allocated are: \n");
        for(i=1;i<k;i++)
        {
                if(store[i]!=-1)
                {
                        printf("p%d: at block-%d\n",i,store[i]);
                }
        }
        int sum=0;
        printf("internal fragment of memory: ");
        for(i=1;i<=m;i++)
        {
                if(visit[i]!=-1)
                {
                        sum=sum+block[i];
                }
        }
        printf("%d\n",sum);
        if(count!=0)
        {
                sum=0;
                for(i=1;i<=m;i++)
                {
                        if(visit[i]==-1){
                                sum=sum+block[i];
                        }
                }
                for(i=0;i<count;i++)
                {
                        if(sum>=size[a[i]])
                        {
                                break;
                        }
                }
                if(i<count)
                {
                        printf("external segment: %d",sum);
                }
                else
                {
                        printf("There is no external fragmentation");
                }
        }
        else
        {
                printf("There is no external fragmentation");
        }
}
```



```
enter the number of process: 3
enter the sizes of process:
process-1 size: 300
process-2 size: 25
process-3 size: 75
enter the number of blocks in memory: 4
Enter the sizes of each blocks:
size of block-1  150
size of block-2  300
size of block-3  30
size of block-4  20
the process allocated are:
p1: at block-2
p2: at block-1
p3: at block-1
internal fragment of memory: 50
There is no external fragmentation
--------------------------------
Process exited after 15.55 seconds with return value 34
Press any key to continue . . .
```

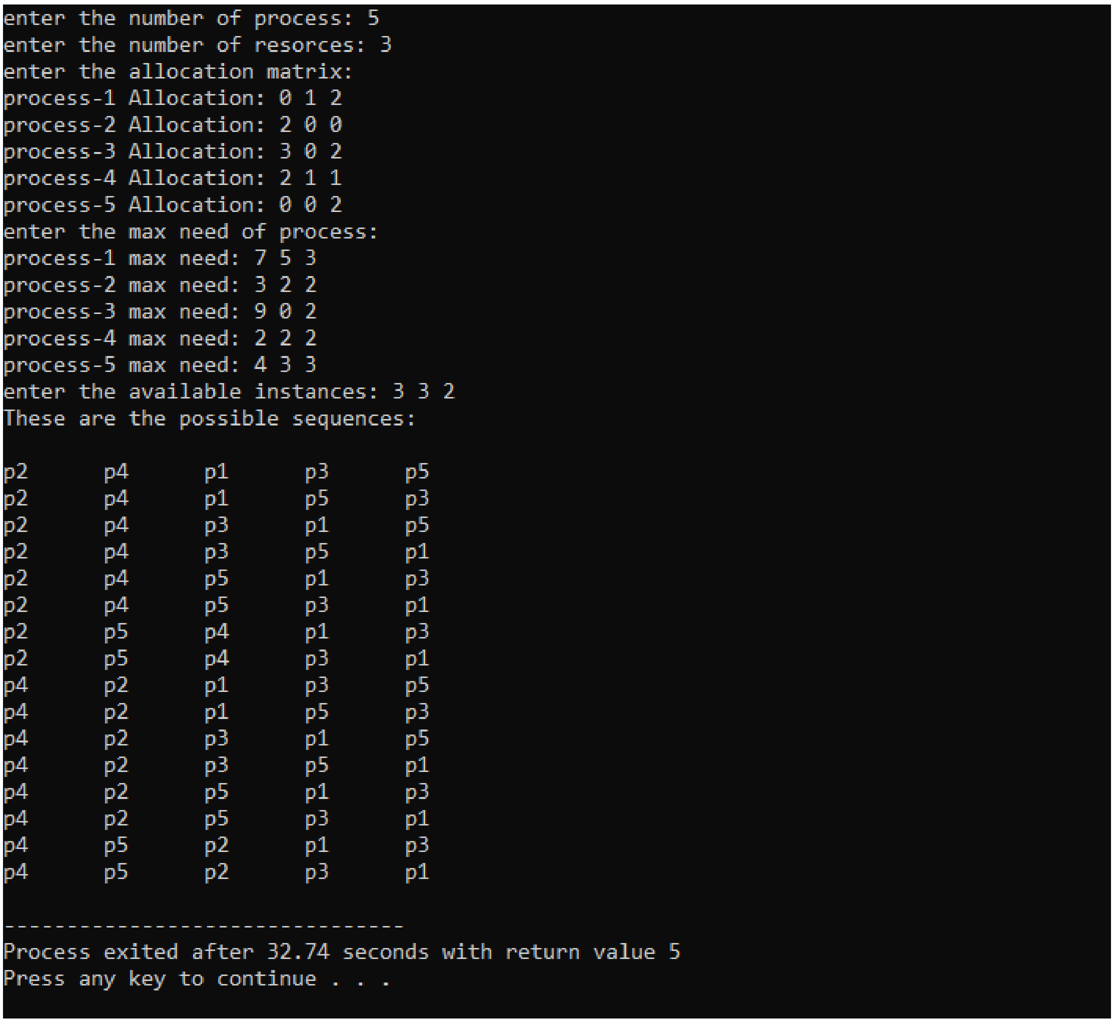
Bankers Algorithm

1. Safe Sequence:

```
#include<stdio.h>
int n,r;
int allocation[100][100],max[100][100],need[100][100],available[100];
int check(int i,int n,int r,int work[])
{
        int j,flag=-1;
        for(j=1;j<=r;j++)
        {
                if(work[j]<need[i][j])
                {
                        flag=1;
                        break;
                }
        }
        return flag;
}
int printing(int sequence[])
{
        int i;
        for(i=1;i<=n;i++)
        {
                printf("p%d\t",sequence[i]);
        }
        printf("\n");
}
int Safety_sequence(int sequence[],int finish[],int work[],int k)
{
        int i,j;
        if(k>n)
        {
                printing(sequence);
                return;
```

```
    }
    for(i=1;i<=n;i++)
    {
        int p;
        p=check(i,n,r,work);
        if(p==-1 && finish[i]==-1)
        {
            sequence[k]=i;
            finish[i]=1;
            for(j=1;j<=r;j++)
            {
                work[j]=work[j]+allocation[i][j];
            }
            Safety_sequence(sequence,finish,work,k+1);
            for(j=1;j<=r;j++)
            {
                work[j]=work[j]-allocation[i][j];
            }
            finish[i]=-1;
        }
    }

}
int main()
{
    int i,j;
    printf("enter the number of process: ");
    scanf("%d",&n);
    printf("enter the number of resorces: ");
    scanf("%d",&r);
    int finish[n],work[r],sequence[n];
    printf("enter the allocation matrix: \n");
    for(i=1;i<=n;i++)
    {
        finish[i]=-1;
        printf("process-%d Allocation: ",i);
        for(j=1;j<=r;j++)
        {
            scanf("%d",&allocation[i][j]);
        }
    }
    printf("enter the max need of process: \n");
    for(i=1;i<=n;i++)
    {
        printf("process-%d max need: ",i);
        for(j=1;j<=r;j++)
        {
            scanf("%d",&max[i][j]);
        }
    }
    for(i=1;i<=n;i++)
```

```
        {
                for(j=1;j<=r;j++)
                {
                        need[i][j]=max[i][j]-allocation[i][j];
                }
        }
        printf("enter the available instances: ");
        int k=1;
        for(i=1;i<=r;i++)
        {
                scanf("%d",&available[i]);
                work[i]=available[i];
        }
        //here starts logic
        printf("These are the possible sequences: \n\n");
        Safety_sequence(sequence,finish,work,k);
}
```



```
enter the number of process: 5
enter the number of resorces: 3
enter the allocation matrix:
process-1 Allocation: 0 1 2
process-2 Allocation: 2 0 0
process-3 Allocation: 3 0 2
process-4 Allocation: 2 1 1
process-5 Allocation: 0 0 2
enter the max need of process:
process-1 max need: 7 5 3
process-2 max need: 3 2 2
process-3 max need: 9 0 2
process-4 max need: 2 2 2
process-5 max need: 4 3 3
enter the available instances: 3 3 2
These are the possible sequences:

p2      p4      p1      p3      p5
p2      p4      p1      p5      p3
p2      p4      p3      p1      p5
p2      p4      p3      p5      p1
p2      p4      p5      p1      p3
p2      p4      p5      p3      p1
p2      p5      p4      p1      p3
p2      p5      p4      p3      p1
p4      p2      p1      p3      p5
p4      p2      p1      p5      p3
p4      p2      p3      p1      p5
p4      p2      p3      p5      p1
p4      p2      p5      p1      p3
p4      p2      p5      p3      p1
p4      p5      p2      p1      p3
p4      p5      p2      p3      p1

--------------------------------
Process exited after 32.74 seconds with return value 5
Press any key to continue . . .
```

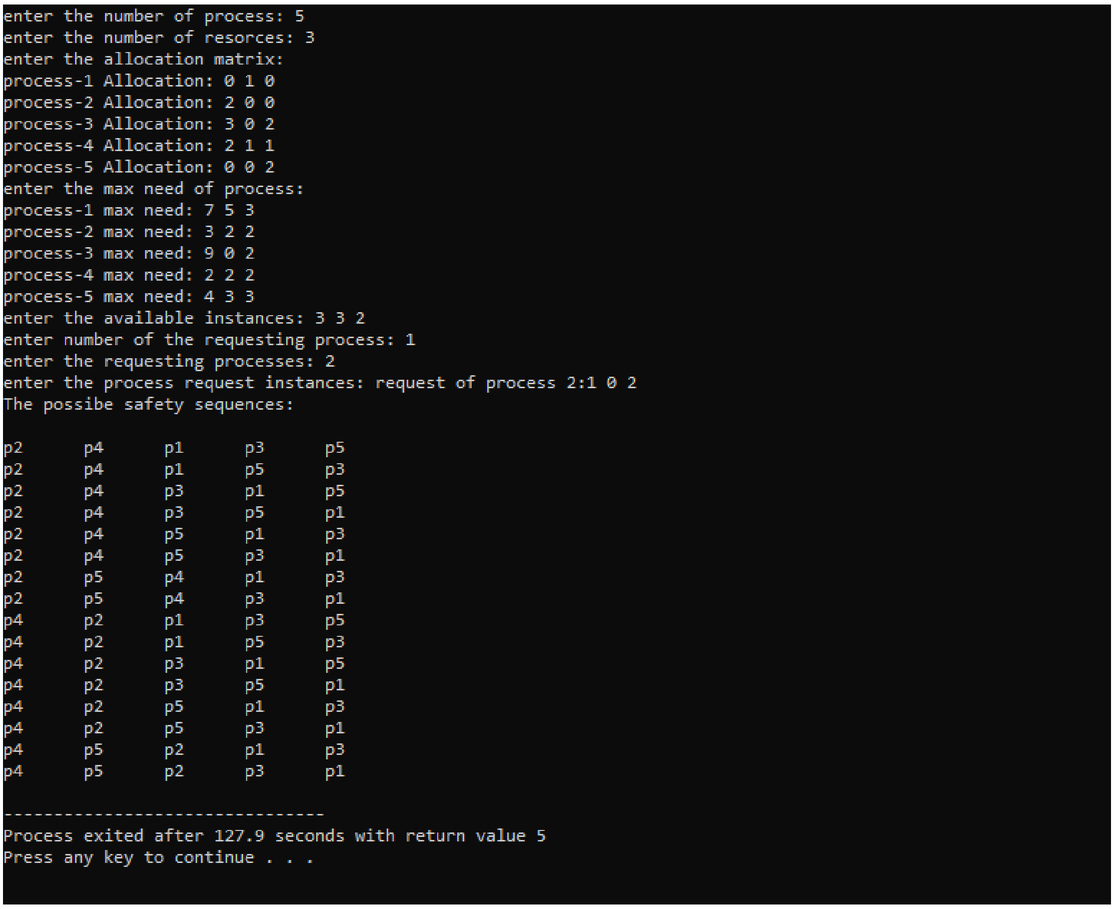
2. Resource Request Algorithm:

```
#include<stdio.h>
int n,r,l=0;
int allocation[100][100],max[100][100],need[100][100],available[100];
int check2(int i,int request[],int req_arr[][r])
{
        int j,flag=1;
        for(j=1;j<=r;j++)
        {
                if(req_arr[i][j]>need[request[i]][j])
                {
                        flag=0;
                        break;
                }
        }
        return flag;
}
int check(int i,int n,int r,int work[])
{
        int j,flag=-1;
        for(j=1;j<=r;j++)
        {
                if(work[j]<need[i][j])
                {
                        flag=1;
                        break;
                }
        }
        return flag;
}
int printing(int sequence[],int k)
{
        int i;
        for(i=1;i<k;i++)
        {
                printf("p%d\t",sequence[i]);
        }
        printf("\n");
}
int Safety_sequence(int sequence[],int finish[],int work[],int k)
{
        int i,j;
        if(k>l+n)
        {
                printing(sequence,k);
                return;
        }
        for(i=1;i<=n;i++)
        {
                int p;
                p=check(i,n,r,work);
                if(p==-1 && finish[i]==-1)
                {
                        sequence[k]=i;
                        finish[i]=1;
```

```
            for(j=1;j<=r;j++)
            {
                work[j]=work[j]+allocation[i][j];
            }
            Safety_sequence(sequence,finish,work,k+1);
            for(j=1;j<=r;j++)
            {
                work[j]=work[j]-allocation[i][j];
            }
            finish[i]=-1;
        }
    }

}
int main()
{
    int i,j;
    printf("enter the number of process: ");
    scanf("%d",&n);
    printf("enter the number of resorces: ");
    scanf("%d",&r);
    int finish[n],work[r],sequence[100];
    printf("enter the allocation matrix: \n");
    for(i=1;i<=n;i++)
    {
        finish[i]=-1;
        printf("process-%d Allocation: ",i);
        for(j=1;j<=r;j++)
        {
            scanf("%d",&allocation[i][j]);
        }
    }
    printf("enter the max need of process: \n");
    for(i=1;i<=n;i++)
    {
        printf("process-%d max need: ",i);
        for(j=1;j<=r;j++)
        {
            scanf("%d",&max[i][j]);
        }
    }
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=r;j++)
        {
            need[i][j]=max[i][j]-allocation[i][j];
        }
    }
    printf("enter the available instances: ");
    int k=1;
    for(i=1;i<=r;i++)
```

```
{
    scanf("%d",&available[i]);
    work[i]=available[i];
}
printf("enter number of the requesting process: ");
int c;
scanf("%d",&c);
int request[c],req_arr[c][r];
printf("enter the requesting processes: ");
for(i=1;i<=c;i++){
    scanf("%d",&request[i]);
}
printf("enter the process request instances: ");
for(i=1;i<=c;i++)
{
    printf("request of process %d:",request[i]);
    for(j=1;j<=r;j++)
    {
        scanf("%d",&req_arr[i][j]);
    }
}
//resource request process alogorithem;
for(i=1;i<=c;i++)
{
    if(check2(i,request,req_arr) && check(i,n,r,work))
    {
        sequence[k]=request[i];
        k++;
        for(j=1;j<=r;j++)
        {
            available[j]-=req_arr[i][j];
            allocation[request[i]][j]+=req_arr[i][j];
            need[request[i]][j]-=req_arr[i][j];
        }
        for(j=1;j<=r;j++)
        {
            if(need[request[i]][j]!=0)
            {
                break;
            }
        }
        if(j>r)
        {
            int t=1;
            for(t=1;t<=r;t++)
            {
                available[t]+=allocation[request[i]][t];
            }
            finish[request[i]]=1;
            l++;
        }
```

```
                }
        }
        //here starts logic
        printf("The possibe safety sequences: \n\n");
        Safety_sequence(sequence,finish,work,k);
}
```

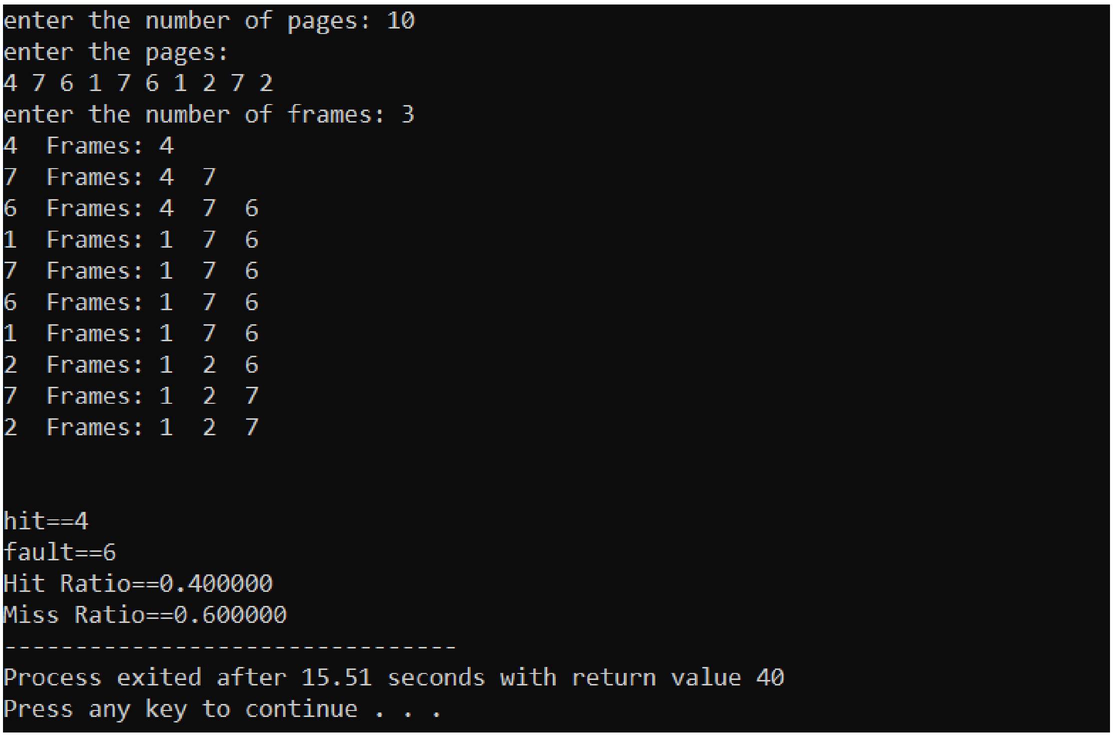


```
enter the number of process: 5
enter the number of resorces: 3
enter the allocation matrix:
process-1 Allocation: 0 1 0
process-2 Allocation: 2 0 0
process-3 Allocation: 3 0 2
process-4 Allocation: 2 1 1
process-5 Allocation: 0 0 2
enter the max need of process:
process-1 max need: 7 5 3
process-2 max need: 3 2 2
process-3 max need: 9 0 2
process-4 max need: 2 2 2
process-5 max need: 4 3 3
enter the available instances: 3 3 2
enter number of the requesting process: 1
enter the requesting processes: 2
enter the process request instances: request of process 2:1 0 2
The possibe safety sequences:

p2      p4      p1      p3      p5
p2      p4      p1      p5      p3
p2      p4      p3      p1      p5
p2      p4      p3      p5      p1
p2      p4      p5      p1      p3
p2      p4      p5      p3      p1
p2      p5      p4      p1      p3
p2      p5      p4      p3      p1
p4      p2      p1      p3      p5
p4      p2      p1      p5      p3
p4      p2      p3      p1      p5
p4      p2      p3      p5      p1
p4      p2      p5      p1      p3
p4      p2      p5      p3      p1
p4      p5      p2      p1      p3
p4      p5      p2      p3      p1

--------------------------------
Process exited after 127.9 seconds with return value 5
Press any key to continue . . .
```

Page Replacement Algorithms

1. FIFO(FIRST IN FIRST OUT)

```
#include<stdio.h>
int pagefound(int page,int frames[],int l)
{
        int i;
        for(i=0;i<l;i++)
        {
                if(frames[i]==page)
                {
                        return 1;
                }
        }
        return  0;
}
void print(int l,int frames[])
{
        int i;
        for(i=0;i<l;i++)
        {
                printf("%d  ",frames[i]);
        }
}
int main()
{
        int n;
        printf("enter the number of pages: ");
        scanf("%d",&n);
        int pages[n];
        printf("enter the pages: \n");
        int i;
        for(i=1;i<=n;i++)
        {
                scanf("%d",&pages[i]);
        }
        printf("enter the number of frames: ");
        int f;
        scanf("%d",&f);
        int frames[f];
        int empty=f;
        int hit=0,fault=0;
        int l=0,top=0;
        printf("\n\n");
        for(i=1;i<=n;i++)
        {
                printf("%d  Frames: ",pages[i]);
                if(empty!=0)
                {
                        if(pagefound(pages[i],frames,l)==0)
                        {
                                frames[l]=pages[i];
                                fault++;
                                l++;
                                empty--;
                        }
                        else
                        {
                                hit++;
                        }
                }
                else{

                        if(pagefound(pages[i],frames,l)==0)
                        {
                                int j;
                                frames[top]=pages[i];
                                fault++;
                                top=(top+1)%f;
                        }
                        else
                        {
                                hit++;
                        }
                }
                print(l,frames);
                printf("\n");
        }
        printf("\n\n");
        printf("hit==%d\nfault==%d",hit,fault);
        printf("\n");
        printf("Hit Ratio==%f\nMiss Ratio==%f",(float)hit/n,(float)fault/n);
}
```

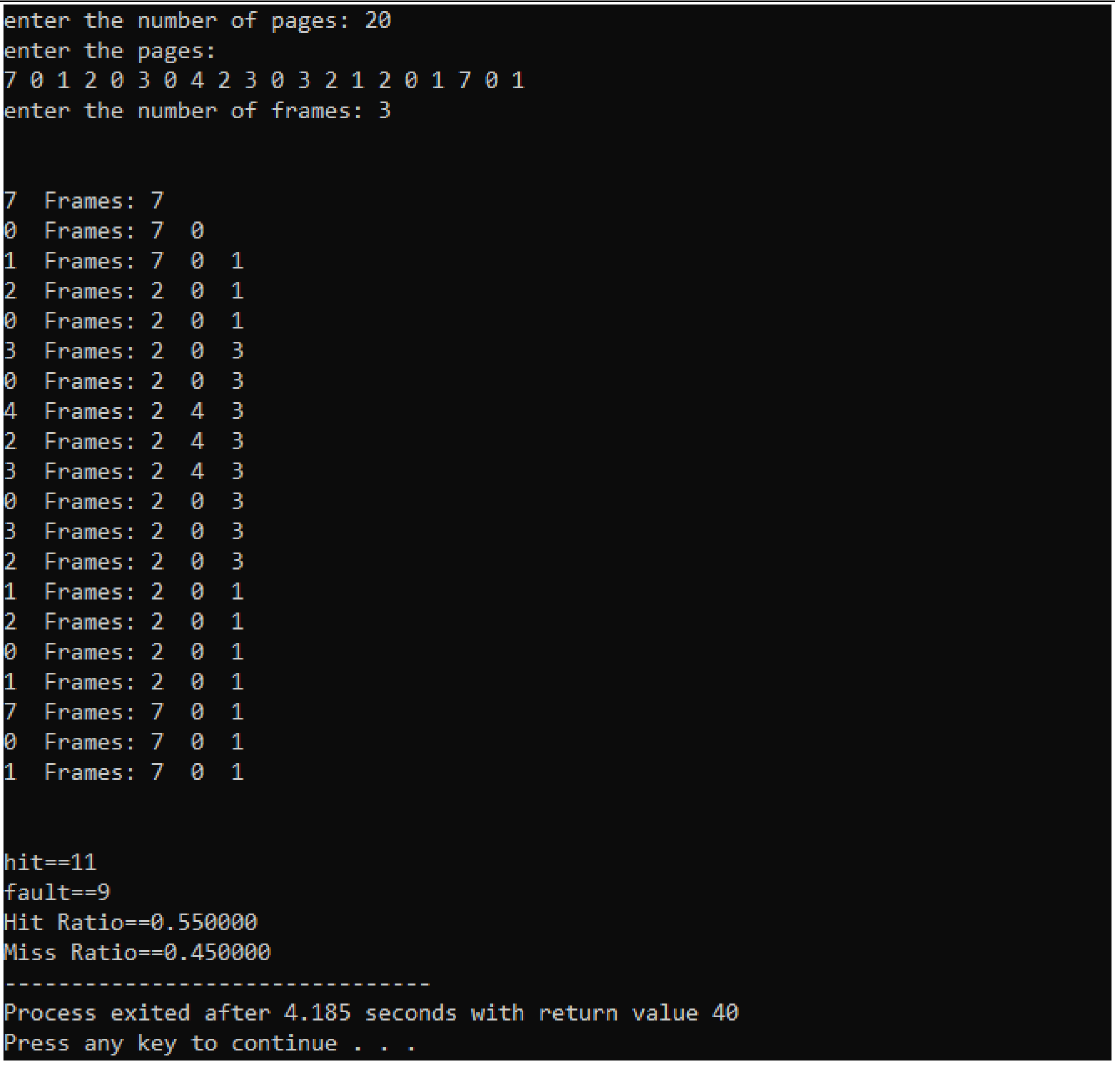


```
enter the number of pages: 10
enter the pages:
4 7 6 1 7 6 1 2 7 2
enter the number of frames: 3
4  Frames: 4
7  Frames: 4  7
6  Frames: 4  7  6
1  Frames: 1  7  6
7  Frames: 1  7  6
6  Frames: 1  7  6
1  Frames: 1  7  6
2  Frames: 1  2  6
7  Frames: 1  2  7
2  Frames: 1  2  7


hit==4
fault==6
Hit Ratio==0.400000
Miss Ratio==0.600000
--------------------------------
Process exited after 15.51 seconds with return value 40
Press any key to continue . . .
```

2. OPTIMAL PAGE REPLACEMENT

```
#include<stdio.h>
int pagefound(int page,int frames[],int l)
{
        int i;
        for(i=0;i<l;i++)
        {
                if(frames[i]==page)
                {
                        return 1;
                }
        }
        return  0;
}
void print(int l,int frames[])
{
        int i;
        for(i=0;i<l;i++)
        {
                printf("%d  ",frames[i]);
        }
}
int check(int i,int pages[],int frame,int n)
{
        int j;
        for(j=i;j<=n;j++)
        {
                if(pages[j]==frame)
                {
                        return j;
                }
        }
        return 999;
}
int main()
{
        int n;
        printf("enter the number of pages: ");
        scanf("%d",&n);
        int pages[n];
        printf("enter the pages: \n");
        int i;
        for(i=1;i<=n;i++)
        {
                scanf("%d",&pages[i]);
        }
        printf("enter the number of frames: ");
        int f;
        scanf("%d",&f);
        int frames[f];
        int empty=f;
        int hit=0,fault=0;
        int l=0;
        printf("\n\n");
        for(i=1;i<=n;i++)
        {
                printf("%d  Frames: ",pages[i]);
                if(empty!=0)
                {
                        if(pagefound(pages[i],frames,l)==0)
                        {
                                frames[l]=pages[i];
                                fault++;
                                l++;
                                empty--;
                        }
                        else
```

```
            {
                hit++;
            }
        }
        else{

            if(pagefound(pages[i],frames,l)==0)
            {
                int visit[f],j;
                for(j=0;j<f;j++)
                {
                    visit[j]=check(i+1,pages,frames[j],n);
                }
                int max=visit[0],q=0;
                for(j=1;j<f;j++)
                {
                    if(max<visit[j])
                    {
                        max=visit[j];
                        q=j;
                    }
                }
                frames[q]=pages[i];
                fault++;
            }
            else
            {
                hit++;
            }
        }
        print(l,frames);
        printf("\n");
    }
    printf("\n\n");
    printf("hit==%d\nfault==%d",hit,fault);
    printf("\n");
    printf("Hit Ratio==%f\nMiss Ratio==%f",(float)hit/n,(float)fault/n);
}
```

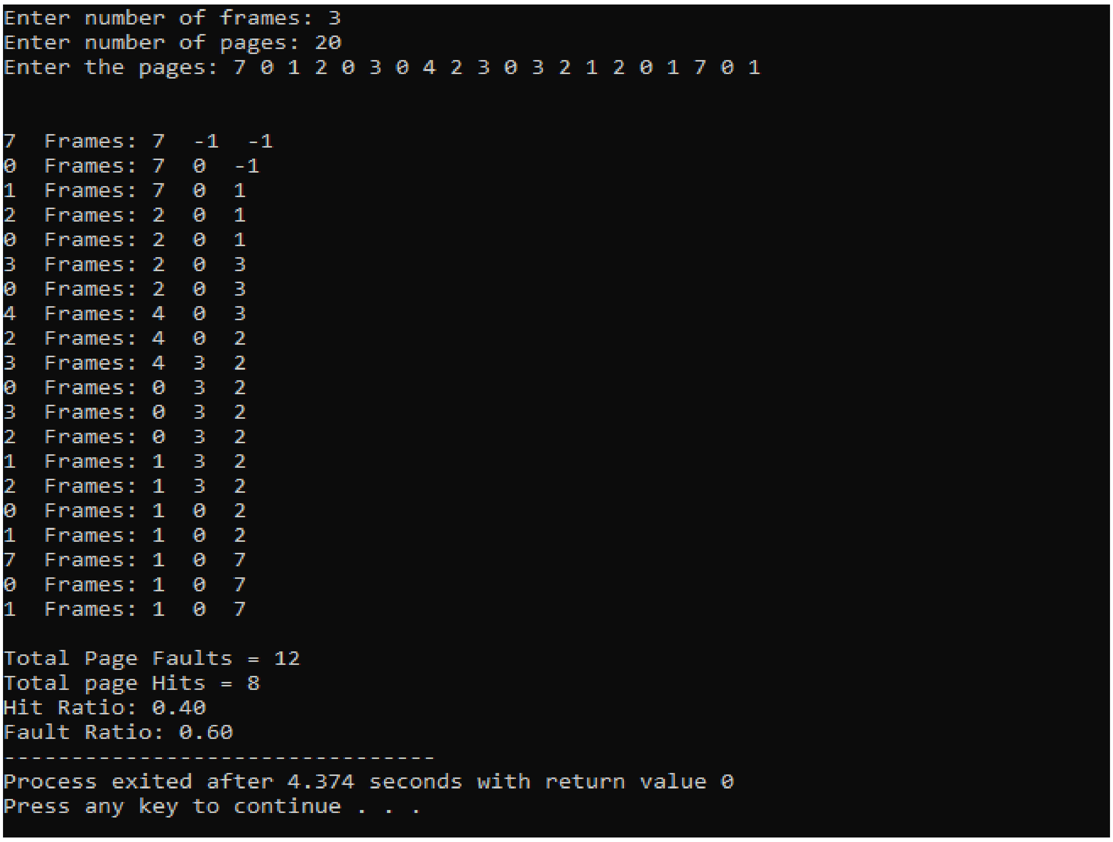


```
enter the number of pages: 20
enter the pages:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
enter the number of frames: 3


7  Frames: 7
0  Frames: 7  0
1  Frames: 7  0  1
2  Frames: 2  0  1
0  Frames: 2  0  1
3  Frames: 2  0  3
0  Frames: 2  0  3
4  Frames: 2  4  3
2  Frames: 2  4  3
3  Frames: 2  4  3
0  Frames: 2  0  3
3  Frames: 2  0  3
2  Frames: 2  0  3
1  Frames: 2  0  1
2  Frames: 2  0  1
0  Frames: 2  0  1
1  Frames: 2  0  1
7  Frames: 7  0  1
0  Frames: 7  0  1
1  Frames: 7  0  1


hit==11
fault==9
Hit Ratio==0.550000
Miss Ratio==0.450000
--------------------------------
Process exited after 4.185 seconds with return value 40
Press any key to continue . . .
```

3. LRU(LEAST RECENTLY USED)

```
#include <stdio.h>

//user-defined function
int findLRU(int time[], int n)
{
  int i, minimum = time[0], pos = 0;

  for (i = 1; i < n; ++i)
  {
    if (time[i] < minimum)
    {
      minimum = time[i];
      pos = i;
    }
  }

  return pos;
}
```

```
//main function
int main()
{
  int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i,
j, pos, faults = 0;
  printf("Enter number of frames: ");
  scanf("%d", &no_of_frames);

  printf("Enter number of pages: ");
  scanf("%d", &no_of_pages);

  printf("Enter the pages: ");

  for (i = 0; i < no_of_pages; ++i)
  {
    scanf("%d", &pages[i]);
  }

  for (i = 0; i < no_of_frames; ++i)
  {
    frames[i] = -1;
  }

          printf("\n\n");
  for (i = 0; i < no_of_pages; ++i)
  {
                    printf("%d  Frames: ",pages[i]);
    flag1 = flag2 = 0;

    for (j = 0; j < no_of_frames; ++j)
    {
      if (frames[j] == pages[i])
      {
        counter++;
        time[j] = counter;
        flag1 = flag2 = 1;
        break;
      }
    }

    if (flag1 == 0)
    {
      for (j = 0; j < no_of_frames; ++j)
      {
        if (frames[j] == -1)
        {
          counter++;
          faults++;
          frames[j] = pages[i];
          time[j] = counter;
          flag2 = 1;
          break;
        }
      }
    }

    if (flag2 == 0)
    {
      pos = findLRU(time, no_of_frames);
      counter++;
      faults++;
      frames[pos] = pages[i];
      time[pos] = counter;
    }
    for (j = 0; j < no_of_frames; ++j)
    {
      printf("%d  ", frames[j]);
    }
    printf("\n");
  }
          int hit=no_of_pages-faults;
  printf("\nTotal Page Faults = %d", faults);
  printf("\nTotal page Hits = %d", hit);
printf("\nHitRatio: %.2f\nFault
Ratio: %.2f",(float)hit/no_of_pages,(float)faults/no_of_pages);

  return 0;
}
```

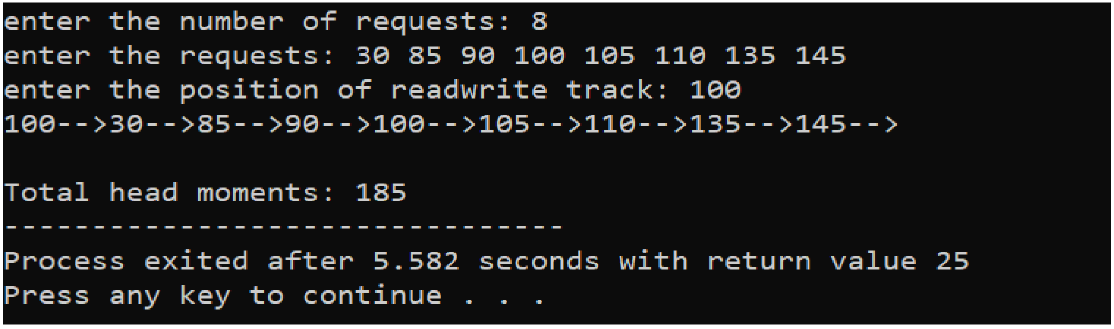


```
Enter number of frames: 3
Enter number of pages: 20
Enter the pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1


7  Frames: 7  -1  -1
0  Frames: 7  0  -1
1  Frames: 7  0  1
2  Frames: 2  0  1
0  Frames: 2  0  1
3  Frames: 2  0  3
0  Frames: 2  0  3
4  Frames: 4  0  3
2  Frames: 4  0  2
3  Frames: 4  3  2
0  Frames: 0  3  2
3  Frames: 0  3  2
2  Frames: 0  3  2
1  Frames: 1  3  2
2  Frames: 1  3  2
0  Frames: 1  0  2
1  Frames: 1  0  2
7  Frames: 1  0  7
0  Frames: 1  0  7
1  Frames: 1  0  7

Total Page Faults = 12
Total page Hits = 8
Hit Ratio: 0.40
Fault Ratio: 0.60
--------------------------------
Process exited after 4.374 seconds with return value 0
Press any key to continue . . .
```

DISK SCHEDULE ALGORITHM

1. FCFS(FIRST COME FIRST SERVE)

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int n,i;
        printf("enter the number of requests: ");
        scanf("%d",&n);
        int request[n];
        printf("enter the requests: ");
        for(i=0;i<n;i++)
        {
                scanf("%d",&request[i]);
        }
        int m;
        printf("enter the position of readwrite track: ");
        scanf("%d",&m);
        printf("%d-->",m);
        for(i=0;i<n;i++)
        {
                printf("%d-->",request[i]);
        }
        int THM=0;
        THM=THM+abs(request[0]-m);
        for(i=1;i<n;i++)
        {
                THM+=abs(request[i]-request[i-1]);
        }
        printf("\n\nTotal head moments: %d",THM);

}
```

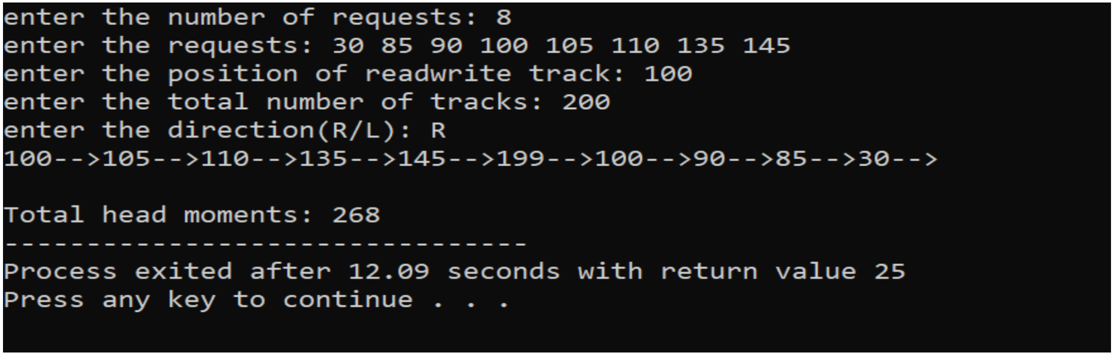


```
enter the number of requests: 8
enter the requests: 30 85 90 100 105 110 135 145
enter the position of readwrite track: 100
100-->30-->85-->90-->100-->105-->110-->135-->145-->

Total head moments: 185
--------------------------------
Process exited after 5.582 seconds with return value 25
Press any key to continue . . .
```

2. SCAN

```
#include<stdio.h>
#include<stdlib.h>
int sort(int arr[],int n)
{
        int i,j;
```

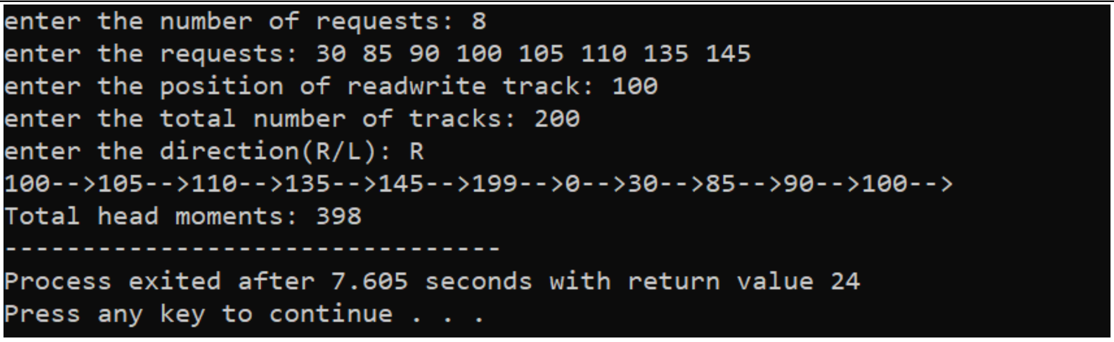
```
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(arr[i]>arr[j])
            {
                int temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
            }
        }
    }
}
int main()
{
    int n,i;
    printf("enter the number of requests: ");
    scanf("%d",&n);
    int request[n+2];
    printf("enter the requests: ");
    for(i=1;i<n+1;i++)
    {
        scanf("%d",&request[i]);
    }
    request[0]=0;
    int m;
    printf("enter the position of readwrite track: ");
    scanf("%d",&m);
    int t;
    printf("enter the total number of tracks: ");
    scanf("%d",&t);
    request[n+1]=t-1;
    sort(request,n+2);
    int index;
    for(i=0;i<n+2;i++)
    {
        if(request[i]>m)
        {
            break;
        }
    }
    index=i;
    printf("enter the direction(R/L): ");
    char c[1];
    scanf("%s",c);
    printf("%d-->",m);
    for(i=index;i<n+2;i++)
    {
        printf("%d-->",request[i]);
    }
    for(i=index-1;i>0;i--)
    {
        printf("%d-->",request[i]);
    }
    int thm=0;
    thm+=abs(request[index]-m);
    for(i=index+1;i<n+2;i++)
    {
        thm+=abs(request[i]-request[i-1]);
    }
    thm+=abs(request[i-1]-request[index-1]);
    for(i=index-2;i>0;i--)
    {
        thm+=abs(request[i]-request[i+1]);
    }
    printf("\n\nTotal head moments: %d",thm);

}
```



```
enter the number of requests: 8
enter the requests: 30 85 90 100 105 110 135 145
enter the position of readwrite track: 100
enter the total number of tracks: 200
enter the direction(R/L): R
100-->105-->110-->135-->145-->199-->100-->90-->85-->30-->

Total head moments: 268
--------------------------------
Process exited after 12.09 seconds with return value 25
Press any key to continue . . .
```

3. C-SCAN

```
#include<stdio.h>
#include<stdlib.h>
int sort(int arr[],int n)
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(arr[i]>arr[j])
            {
                int temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
            }
        }
    }
}
int main()
{
    int n,i;
    printf("enter the number of requests: ");
    scanf("%d",&n);
    int request[n+2];
    printf("enter the requests: ");
    for(i=1;i<n+1;i++)
    {
        scanf("%d",&request[i]);
    }
    request[0]=0;
    int m;
    printf("enter the position of readwrite track: ");
    scanf("%d",&m);
    int t;
    printf("enter the total number of tracks: ");
    scanf("%d",&t);
    request[n+1]=t-1;
    sort(request,n+2);;
    int index;
    for(i=0;i<n+2;i++)
    {
        if(request[i]>m)
        {
            break;
        }
    }
    index=i;
    printf("enter the direction(R/L): ");
    char c[1];
    scanf("%s",c);
    printf("%d-->",m);
    for(i=index;i<n+2;i++)
    {
        printf("%d-->",request[i]);
    }
    for(i=0;i<index;i++)
    {
        printf("%d-->",request[i]);
    }
    int thm=0;
    thm+=abs(request[index]-m);
    for(i=index+1;i<n+2;i++)
    {
        thm+=abs(request[i]-request[i-1]);
    }
    thm+=abs(request[i-1]-0);
    for(i=0;i<index-1;i++)
    {
        thm+=abs(request[i]-request[i+1]);
    }
    printf("\nTotal head moments: %d",thm);

}
```

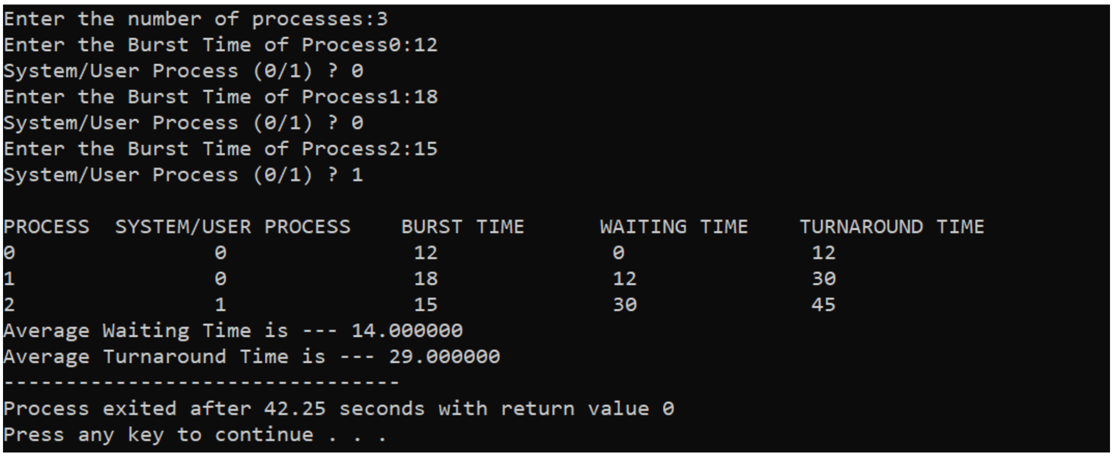


```
enter the number of requests: 8
enter the requests: 30 85 90 100 105 110 135 145
enter the position of readwrite track: 100
enter the total number of tracks: 200
enter the direction(R/L): R
100-->105-->110-->135-->145-->199-->0-->30-->85-->90-->100-->
Total head moments: 398
--------------------------------
Process exited after 7.605 seconds with return value 24
Press any key to continue . . .
```

Multilevel Queue:

```
#include<stdio.h>
int main()
{
        int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp;
        float wtavg, tatavg;
        printf("Enter the number of processes:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                p[i] = i;
                printf("Enter the Burst Time of Process%d:", i);
                scanf("%d",&bt[i]);
                printf("System/User Process (0/1) ? ");
                scanf("%d", &su[i]);
}
        for(i=0;i<n;i++)
                for(k=i+1;k<n;k++)
                        if(su[i] > su[k])
                        {
                        temp=p[i];
                        p[i]=p[k];
                        p[k]=temp;
                        temp=bt[i];
                        bt[i]=bt[k];
                        bt[k]=temp;
                        temp=su[i];
                        su[i]=su[k];
                        su[k]=temp;
                        }
        wtavg = wt[0] = 0;
        tatavg = tat[0] = bt[0];
        for(i=1;i<n;i++)
        {
                wt[i] = wt[i-1] + bt[i-1];
                tat[i] = tat[i-1] + bt[i];
                wtavg = wtavg + wt[i];
                tatavg = tatavg + tat[i];
```

```
	}
	printf("\nPROCESS\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
	for(i=0;i<n;i++)
		printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],su[i],bt[i],wt[i],tat[i]);
	printf("\nAverage Waiting Time is --- %f",wtavg/n);
	printf("\nAverage Turnaround Time is --- %f",tatavg/n);
	return 0;
}
```

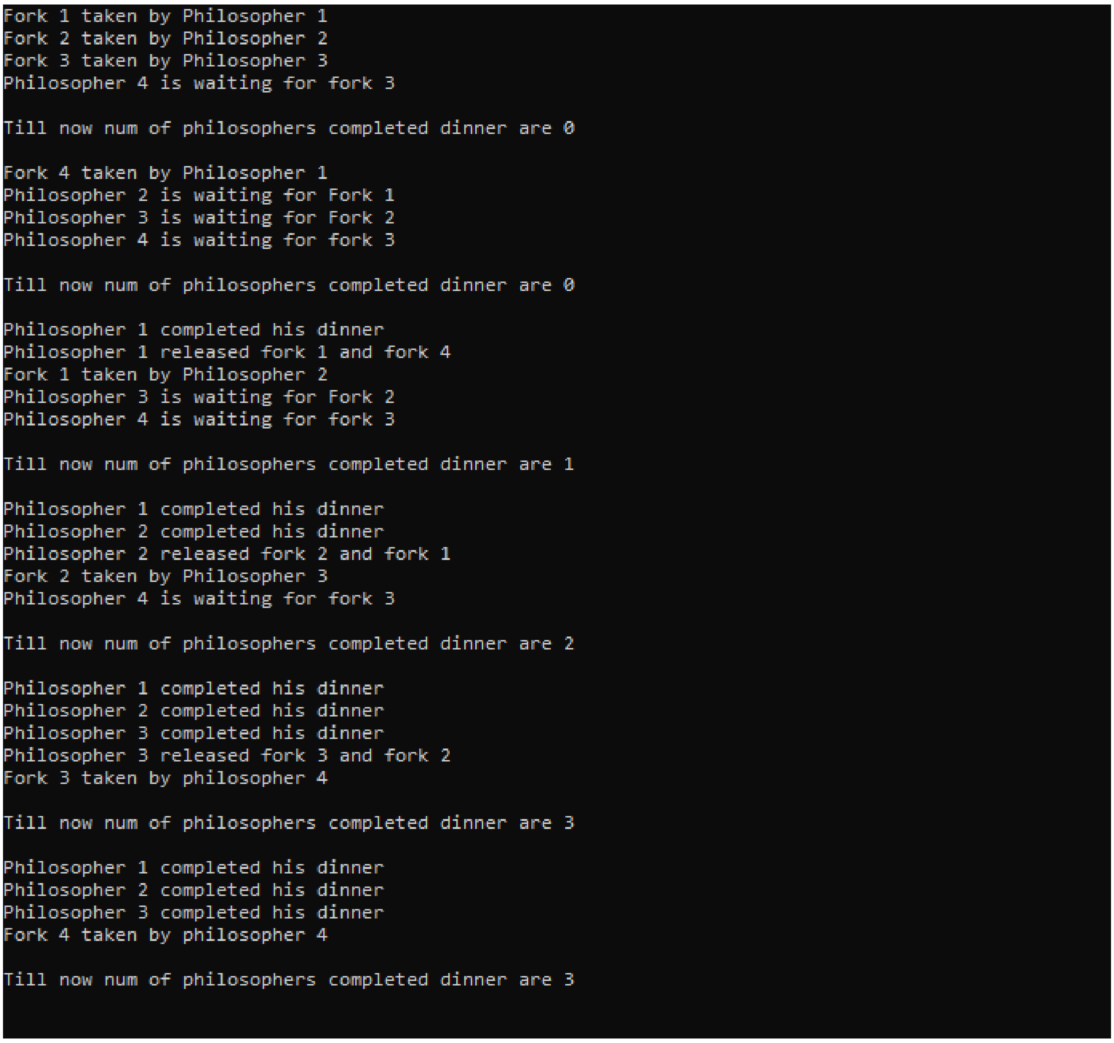


```
Enter the number of processes:3
Enter the Burst Time of Process0:12
System/User Process (0/1) ? 0
Enter the Burst Time of Process1:18
System/User Process (0/1) ? 0
Enter the Burst Time of Process2:15
System/User Process (0/1) ? 1

PROCESS  SYSTEM/USER PROCESS    BURST TIME      WAITING TIME    TURNAROUND TIME
0                0               12              0               12
1                0               18              12              30
2                1               15              30              45
Average Waiting Time is --- 14.000000
Average Turnaround Time is --- 29.000000
--------------------------------
Process exited after 42.25 seconds with return value 0
Press any key to continue . . .
```

Dining Philosoher:

```
#include<stdio.h>

#define n 4

int compltedPhilo = 0,i;

struct fork{
	int taken;
}ForkAvil[n];

struct philosp{
	int left;
	int right;
}Philostatus[n];

void goForDinner(int philID){
	if(Philostatus[philID].left==10 && Philostatus[philID].right==10)
      printf("Philosopher %d completed his dinner\n",philID+1);
	else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){
        printf("Philosopher %d completed his dinner\n",philID+1);

        Philostatus[philID].left = Philostatus[philID].right = 10;
        int otherFork = philID-1;
      if(otherFork== -1)
        otherFork=(n-1);

      ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;
      printf("Philosopher %d released fork %d and
fork %d\n",philID+1,philID+1,otherFork+1);
      compltedPhilo++;
    }
    else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){
        if(philID==(n-1)){
          if(ForkAvil[philID].taken==0){
            ForkAvil[philID].taken = Philostatus[philID].right = 1;
            printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);
          }else{
            printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);
          }
        }else{
          int dupphilID = philID;
          philID-=1;

          if(philID== -1)
            philID=(n-1);

          if(ForkAvil[philID].taken == 0){
            ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;
            printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);
          }else{
            printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);
          }
        }
      }
      else if(Philostatus[philID].left==0){
          if(philID==(n-1)){
            if(ForkAvil[philID-1].taken==0){
              ForkAvil[philID-1].taken = Philostatus[philID].left = 1;
              printf("Fork %d taken by philosopher %d\n",philID,philID+1);
            }else{
              printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
            }
          }else{
            if(ForkAvil[philID].taken == 0){
              ForkAvil[philID].taken = Philostatus[philID].left = 1;
              printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
            }else{
              printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
            }
          }
    }else{}
}
```

```
int main(){
        for(i=0;i<n;i++)
    ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;

        while(compltedPhilo<n){
                for(i=0;i<n;i++)
      goForDinner(i);
                printf("\nTill now num of philosophers completed dinner
are %d\n\n",compltedPhilo);
        }

        return 0;
}
```



```
Fork 1 taken by Philosopher 1
Fork 2 taken by Philosopher 2
Fork 3 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Fork 4 taken by Philosopher 1
Philosopher 2 is waiting for Fork 1
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0

Philosopher 1 completed his dinner
Philosopher 1 released fork 1 and fork 4
Fork 1 taken by Philosopher 2
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 1

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 2 released fork 2 and fork 1
Fork 2 taken by Philosopher 3
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 2

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 3 released fork 3 and fork 2
Fork 3 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Fork 4 taken by philosopher 4

Till now num of philosophers completed dinner are 3
```