FIFO Scheduling

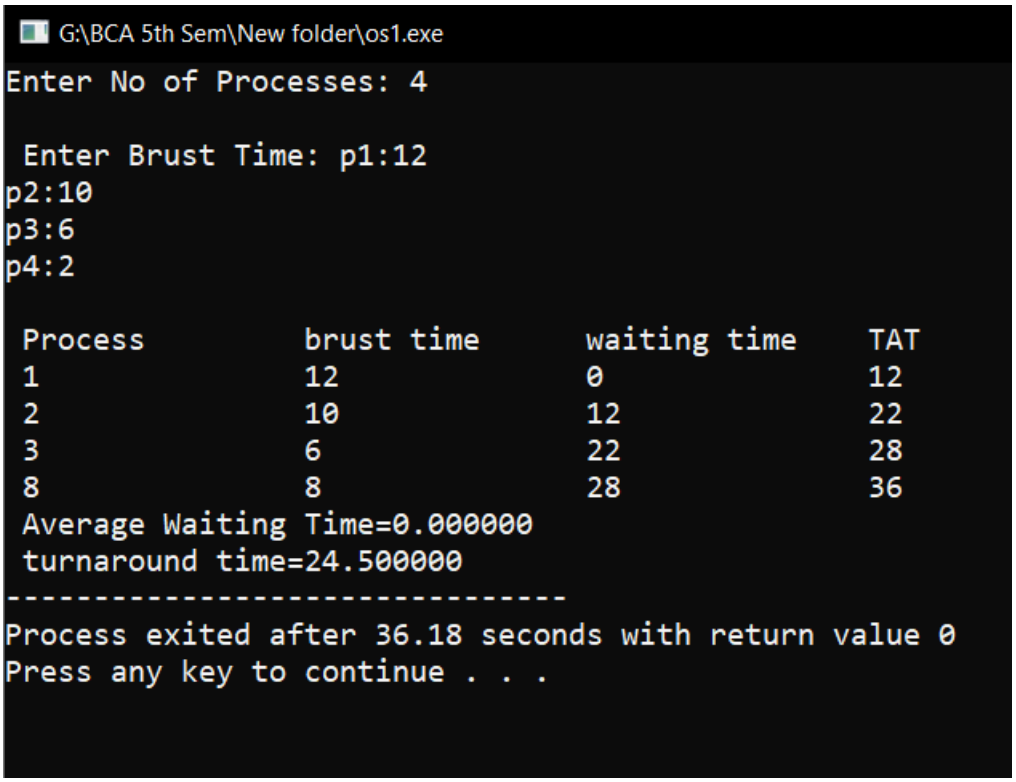
```
#include<stdio.h>
int main()
{
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("Enter total number of processes(maximum 20):"); scanf("%d",&n);
   printf("\nEnter Process Burst Time\n");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&bt[i]);
   }
   wt[0]=0;
   //calculating waiting time
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];
   }
   printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
   //calculating turnaround time
   for(i=0;i<n;i++)
   {
      tat[i]=bt[i]+wt[i];
      avwt+=wt[i];
      avtat+=tat[i];
      printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]); }
   avwt/=i;
   avtat/=i;
   printf("\n\nAverage Waiting Time:%d",avwt);
   printf("\nAverage Turnaround Time:%d",avtat);
   return 0;
   }
```

```
G:\BCA 5th Sem\New folder\os1.exe
Enter total number of processes(maximum 20):4

Enter Process Burst Time
P[1]:12
P[2]:6
P[3]:2
P[4]:1

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            12              0               12
P[2]            6               12              18
P[3]            2               18              20
P[4]            1               20              21

Average Waiting Time:12
Average Turnaround Time:17
--------------------------------
Process exited after 34.19 seconds with return value 0
Press any key to continue . . .
```

SJF Scheduling.

```
#include<stdio.h>
int main()
{
        int bt[20], p[20], wt[20], tat[20], i,j,n, total=0, pos,temp;
        float avg_wt, avg_tat;
        printf("Enter No of Processes");
        scanf("%d", &n);
        printf("\n Enter Brust Time");
        for(i=0; i<n; i++)
        {
                printf("p%d:",i+1);
                scanf("%d", &bt[i]);
                p[i]=i+1;
        }
        for(i=0; i<n; i++)
        {
                pos=i;
                for(j=i+1; j<n; j++);
        }
        if(bt[j]<bt[pos])
        pos=j;
        {
                temp=bt[i];
                bt[i]=bt[pos];
                bt[pos]=temp;
                p[i]=p[pos];
                p[pos]=temp;
        }
        wt[0]=0;
        for(i=1; i<n; i++)
        {
                wt[i]=0;
                for(j=0; j<i; j++)
                wt[i]+=bt[j];
                total+=wt[i];
        }
        avg_tat=(float)total/n;
        total=0;
        printf("\n Process \t brust time \t waiting time \t TAT");
        for(i=0; i<n; i++)
        {
```

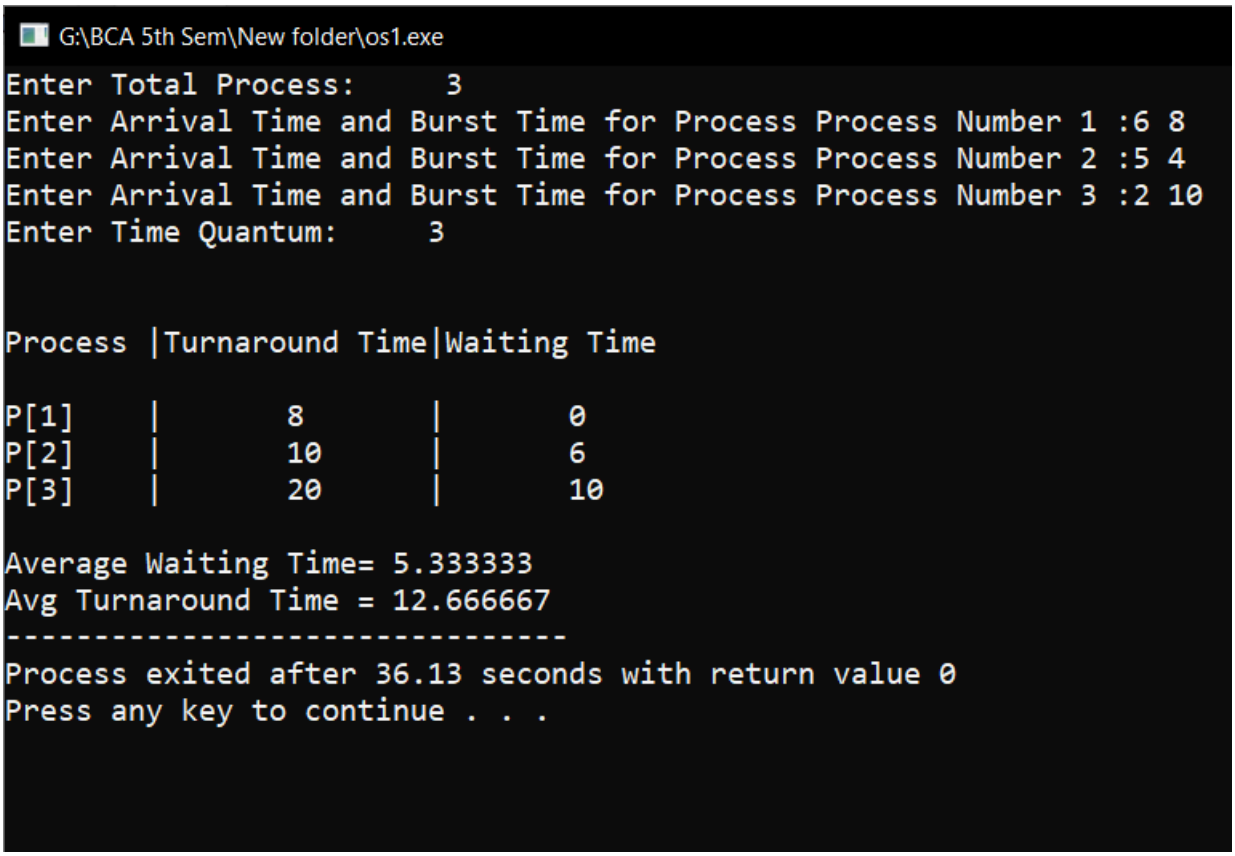
```
            tat[i]=bt[i]+wt[i];
            total+=tat[i];
            printf("\n %d\t\t %d\t\t %d\t\t %d", p[i],bt[i], wt[i],tat[i]); }
        avg_tat= (float)total/n;
        printf("\n Average Waiting Time=%f",avg_wt);
        printf("\n turnaround time=%f",avg_tat);

    }
```



Round-Robin Scheduling.

```
#include<stdio.h>

int main()
{

  int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process
Number %d :",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
    rt[count]=bt[count];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d",&time_quantum);
  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
  for(time=0,count=0;remain!=0;)
  {
    if(rt[count]<=time_quantum && rt[count]>0)
    {
      time+=rt[count];
      rt[count]=0;
      flag=1;
    }
    else if(rt[count]>0)
    {
      rt[count]-=time_quantum;
      time+=time_quantum;
    }
    if(rt[count]==0 && flag==1)
    {
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-
bt[count]);
      wait_time+=time-at[count]-bt[count];
      turnaround_time+=time-at[count];
```

```
    flag=0;
   }
   if(count==n-1)
    count=0;
   else if(at[count+1]<=time)
    count++;
   else
    count=0;
  }
  printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
  printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

  return 0;
}
```



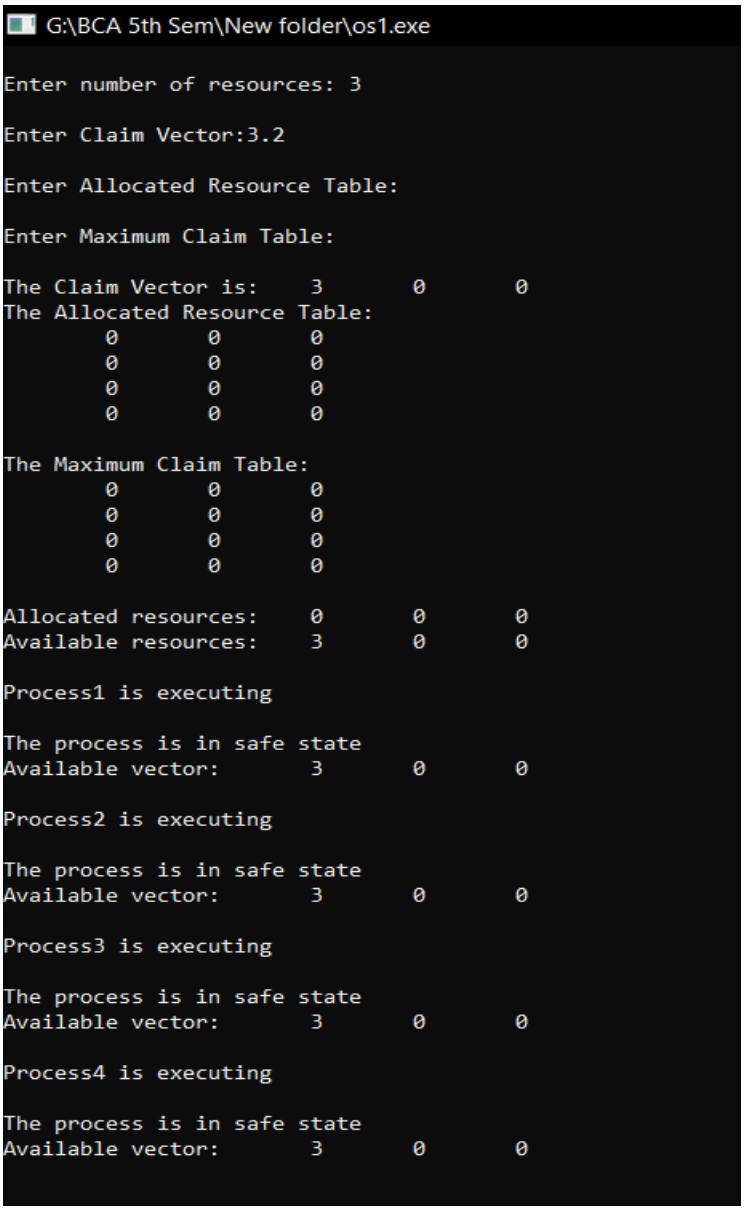
Banker's Algorithm Simulation.

```
#include <stdio.h>

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

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

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

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

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

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

    printf("\nEnter Maximum Claim Table:\n"); for (i = 0; i <
    processes; i++)
{
       for(j = 0; j < resources; j++)
{
```

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

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

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

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

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

```
      available[i] = maxres[i] - allocation[i];
}

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

   while (counter != 0)
{
      safe = 0;
      for (i = 0; i < processes; i++)
{
         if (running[i])
{
            exec = 1;
            for (j = 0; j < resources; j++)
{
               if (maximum_claim[i][j] - current[i][j] > available[j]) {
                  exec = 0;
                  break;
               }
            }
            if (exec)
{
               printf("\nProcess%d is executing\n", i + 1);
               running[i] = 0;
               counter--;
               safe = 1;

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

```
      }
else
{
     printf("\nThe process is in safe state");
          printf("\nAvailable vector:");

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

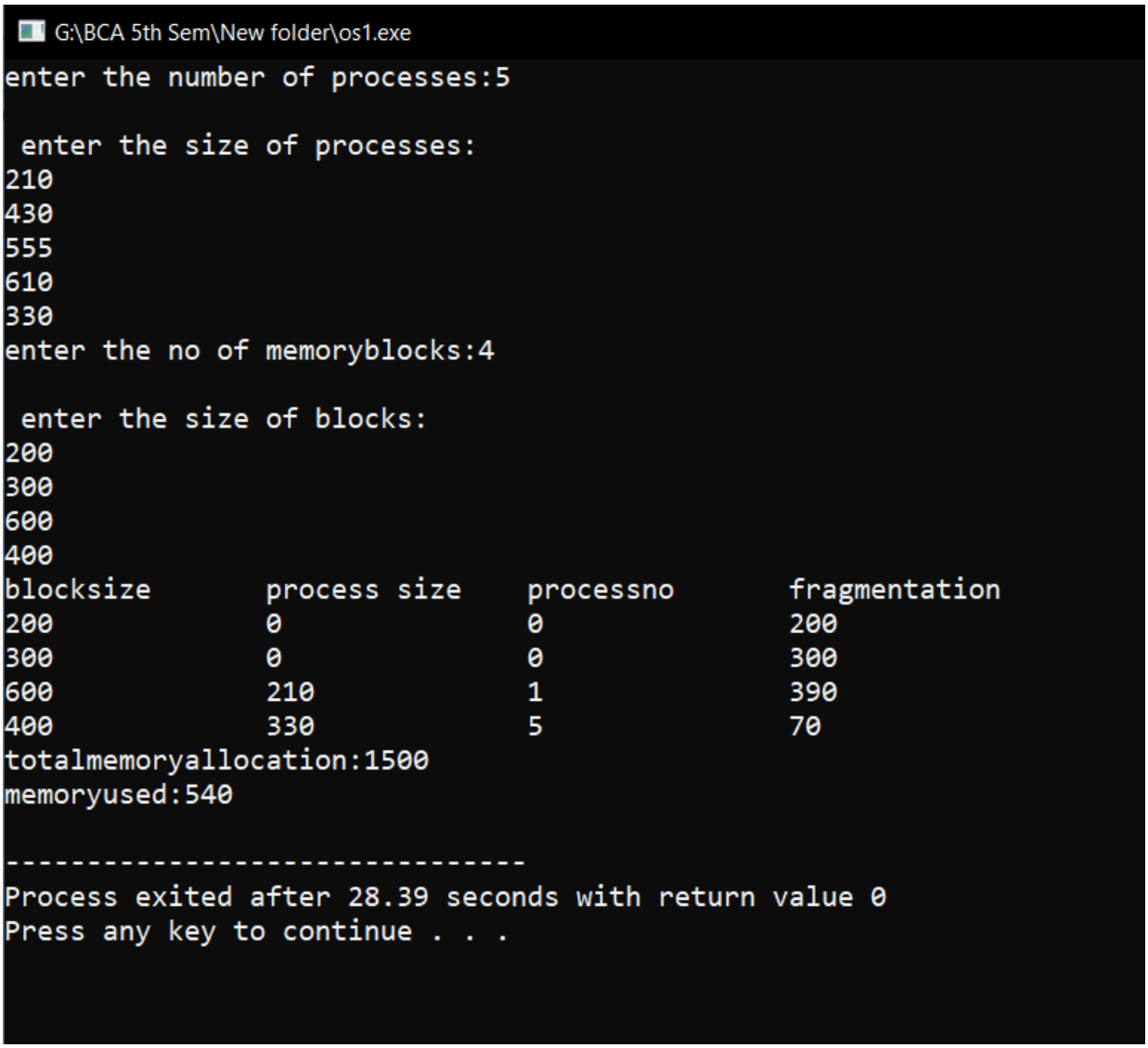


Continuous Memory Allocation Techniques.

a) Worst Fit

```
#include<stdio.h>
int main()
{
int n,n1,i;
printf("enter the number of processes:");
scanf("%d",&n);
int process[n];
printf("\n enter the size of processes:\n");
for(i=0;i<n;i++)
{
scanf("%d",&process[i]);
}
printf("enter the no of memoryblocks:");
scanf("%d",&n1);
int blocks[n1];
printf("\n enter the size of blocks:\n");
int total=0;
for(i=0;i<n1;i++)
{
scanf("%d",&blocks[i]);
total=total+blocks[i];
}
int process1[n1];
int job[n1];
int frag[n1];
int check[n1];
for(i=0;i<n1;i++)
{
check[i]=0;
}
int j,used=0;
i=0;
while(i<n)
{
int max=-1,j1=-1,k=-1,max1;
for(j=0;j<n1;j++)
{
max1=blocks[j];
```

```
if(max1>=max&&check[j]==0&&max1>=process[i]) {
max=max1;
j1=j;
}
else
{
if(check[j]==0)
{
process1[j]=0;
job[j]=0;
frag[j]=blocks[j];
}
}
}
if(k!=j1)
{
process1[j1]=process[i];
job[j1]=i+1;
frag[j1]=blocks[j1]-process[i];
used=used+process[i];
check[j1]=1;
int l;
}
i++;
}
printf("blocksize\tprocess size\tprocessno\tfragmentation\n");
for(i=0;i<n1;i++)
{
printf("%d\t\t%d\t\t%d\t\t%d\n",blocks[i],process1[i],job[i],frag[i]); }
printf("totalmemoryallocation:%d\n",total);
printf("memoryused:%d\n",used);
}
```


G:\BCA 5th Sem\New folder\os1.exe
enter the number of processes:5
enter the size of processes:
210
430
555
610
330
enter the no of memoryblocks:4
enter the size of blocks:
200
300
600
400
blocksize process size processno fragmentation
200 0 0 200
300 0 0 300
600 210 1 390
400 330 5 70
totalmemoryallocation:1500
memoryused:540
Process exited after 28.39 seconds with return value 0
Press any key to continue . . .

b) Best Fit

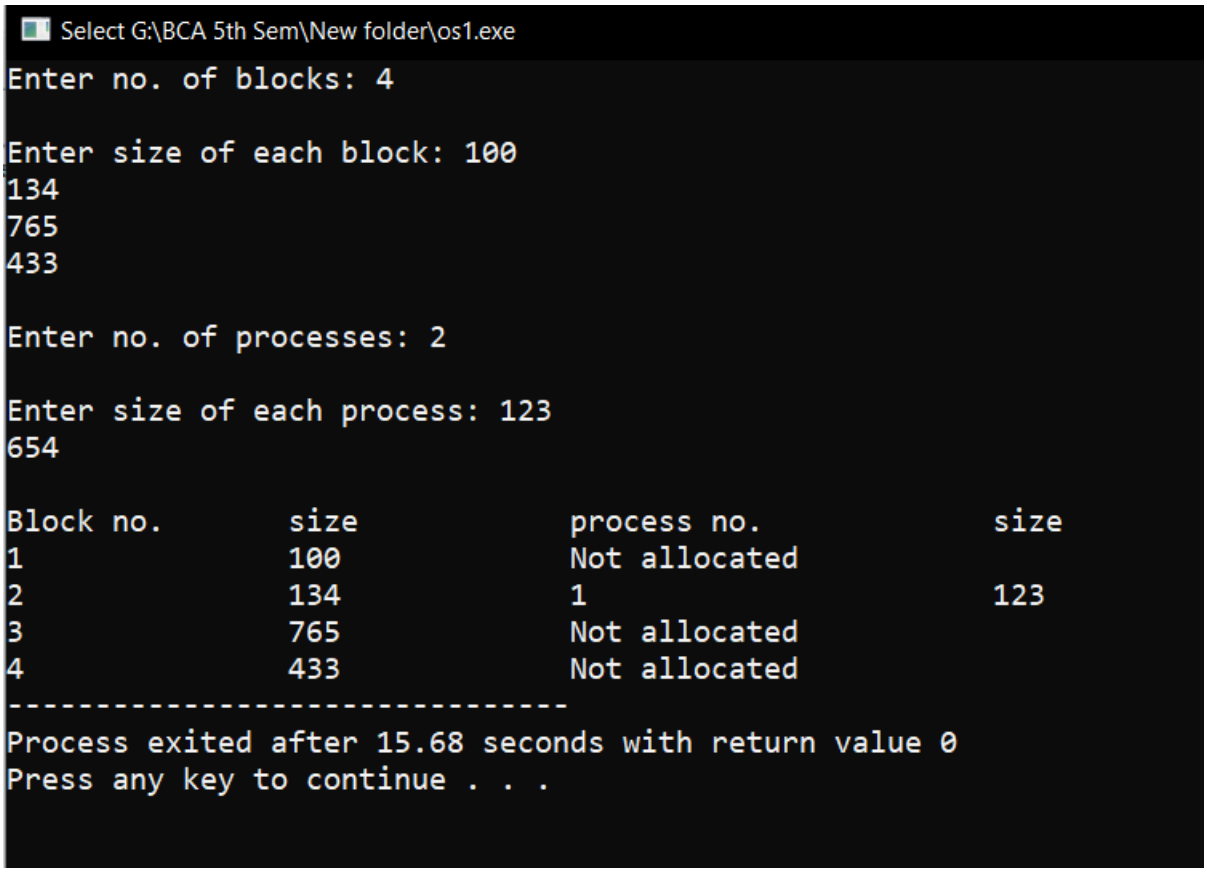
```
#include<stdio.h>

int main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20];
printf("\n\t\t\tMemory Management Scheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of processes:");
scanf("%d",&np);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
   {
printf("Block no.%d:",i);
      scanf("%d",&b[i]);
   }
printf("\nEnter the size of the processes :-\n");
for(i=1;i<=np;i++)
   {
      printf("Process no.%d:",i);
      scanf("%d",&p[i]);
   }
for(i=1;i<=np;i++)
{
for(j=1;j<=nb;j++)
{
if(barray[j]!=1)
{
temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp)
{
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
```

```
                lowest=10000;
                }
                printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
                for(i=1;i<=np && parray[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
  return 0;
    }
```

```
G:\BCA 5th Sem\New folder\os1.exe

Enter the number of blocks:5
Enter the number of processes:4

Enter the size of the blocks:-
Block no.1:200
Block no.2:120
Block no.3:600
Block no.4:420
Block no.5:330

Enter the size of the processes :-
Process no.1:388
Process no.2:122
Process no.3:653
Process no.4:342

Process_no      Process_size    Block_no        Block_size      Fragment
1               388             4               420             32
2               122             1               200             78
--------------------------------
Process exited after 30.86 seconds with return value 0
Press any key to continue . . .
```

c) First Fit

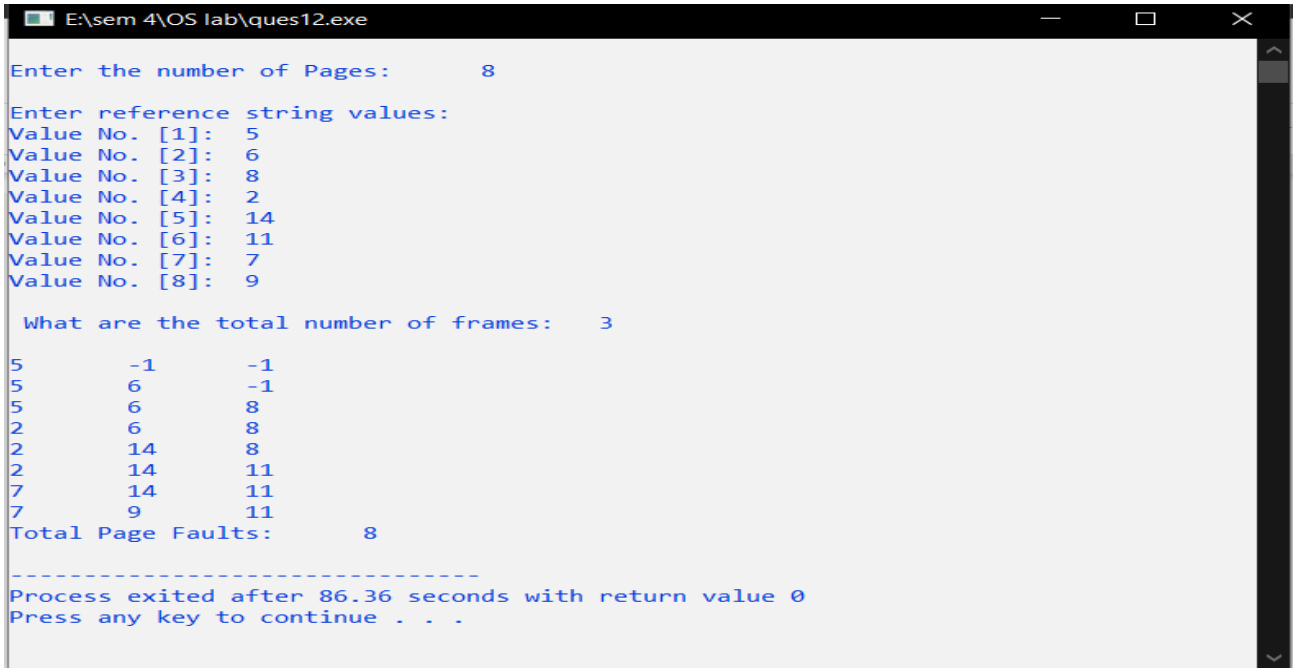
```
#include<stdio.h>

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

for(i = 0; i < 10; i++)
{
flags[i] = 0;
allocation[i] = -1;
}
printf("Enter no. of blocks: ");
scanf("%d", &bno);
printf("\nEnter size of each block: ");
for(i = 0; i < bno; i++)
scanf("%d", &bsize[i]);

printf("\nEnter no. of processes: ");
scanf("%d", &pno);
printf("\nEnter size of each process: ");
for(i = 0; i < pno; i++)
scanf("%d", &psize[i]);
for(i = 0; i < pno; i++) //allocation as per first fit for(j = 0;
j < bno; j++)
if(flags[j] == 0 && bsize[j] >= psize[i])
{
allocation[j] = i;
flags[j] = 1;
break;
}
//display allocation details
printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
for(i = 0; i < bno; i++)
{
printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
if(flags[i] == 1)
printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
else
printf("Not allocated");
}
}
```


Select G:\BCA 5th Sem\New folder\os1.exe
Enter no. of blocks: 4
Enter size of each block: 100
134
765
433
Enter no. of processes: 2
Enter size of each process: 123
654
Block no. size process no. size
1 100 Not allocated
2 134 1 123
3 765 Not allocated
4 433 Not allocated
Process exited after 15.68 seconds with return value 0
Press any key to continue . . .

FIFO Page Replacement.

```
#include <stdio.h>
int main()
{
int referenceString[10], pageFaults = 0, m, n, s, pages, frames;
printf("\nEnter the number of Pages:\t");
scanf("%d", &pages);
printf("\nEnter reference string values:\n");
for( m = 0; m < pages; m++)
{
  printf("Value No. [%d]:\t", m + 1); scanf("%d",
  &referenceString[m]);
}
printf("\n What are the total number of frames:\t"); {
  scanf("%d", &frames);
}
int temp[frames];
for(m = 0; m < frames; m++)
{
 temp[m] = -1;
}
for(m = 0; m < pages; m++)
{
 s = 0;
 for(n = 0; n < frames; n++)
  {
    if(referenceString[m] == temp[n])
      {
        s++;
        pageFaults--;
      }
  }
  pageFaults++;
if((pageFaults <= frames) && (s == 0)) {
      temp[m] = referenceString[m];
    }
  else if(s == 0)
    {
     temp[(pageFaults - 1) % frames] = referenceString[m]; }
    printf("\n");
    for(n = 0; n < frames; n++)
     {
```

```
            printf("%d\t", temp[n]);
        }
    }
    printf("\nTotal Page Faults:\t%d\n", pageFaults);
    return 0;
    }
```

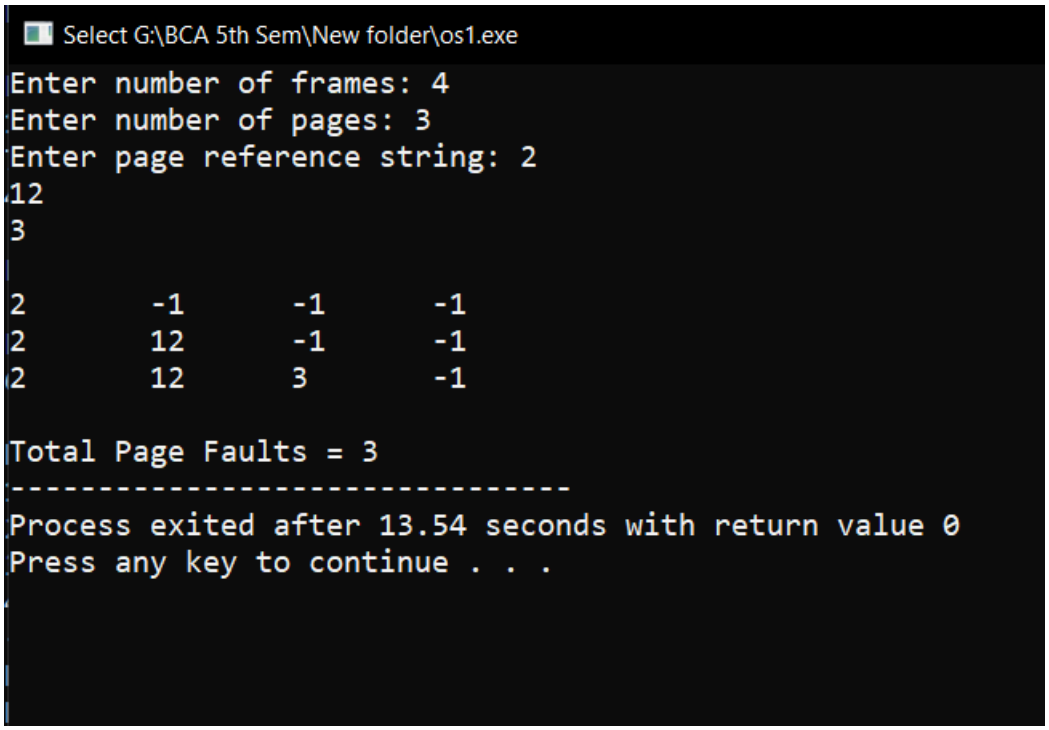


Optimal Page Replacement.

```
#include<stdio.h>
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2,
flag3, i, j, k, pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);

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

   printf("Enter page reference string: ");

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

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

   for(i = 0; i < no_of_pages; ++i){
      flag1 = flag2 = 0;

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

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

```
        if(flag2 == 0){
        flag3 =0;
          for(j = 0; j < no_of_frames; ++j){
           temp[j] = -1;

            for(k = i + 1; k < no_of_pages; ++k){
            if(frames[j] == pages[k]){
            temp[j] = k;
            break;
            }
            }
           }

          for(j = 0; j < no_of_frames; ++j){
           if(temp[j] == -1){
            pos = j;
            flag3 = 1;
            break;
            }
           }

           if(flag3 ==0){
            max = temp[0];
            pos = 0;

            for(j = 1; j < no_of_frames; ++j){
            if(temp[j] > max){
            max = temp[j];
            pos = j;
            }
            }
           }
frames[pos] = pages[i];
faults++;
       }

       printf("\n");

          for(j = 0; j < no_of_frames; ++j){
               printf("%d\t", frames[j]);
       }
    }

    printf("\n\nTotal Page Faults = %d", faults);
```

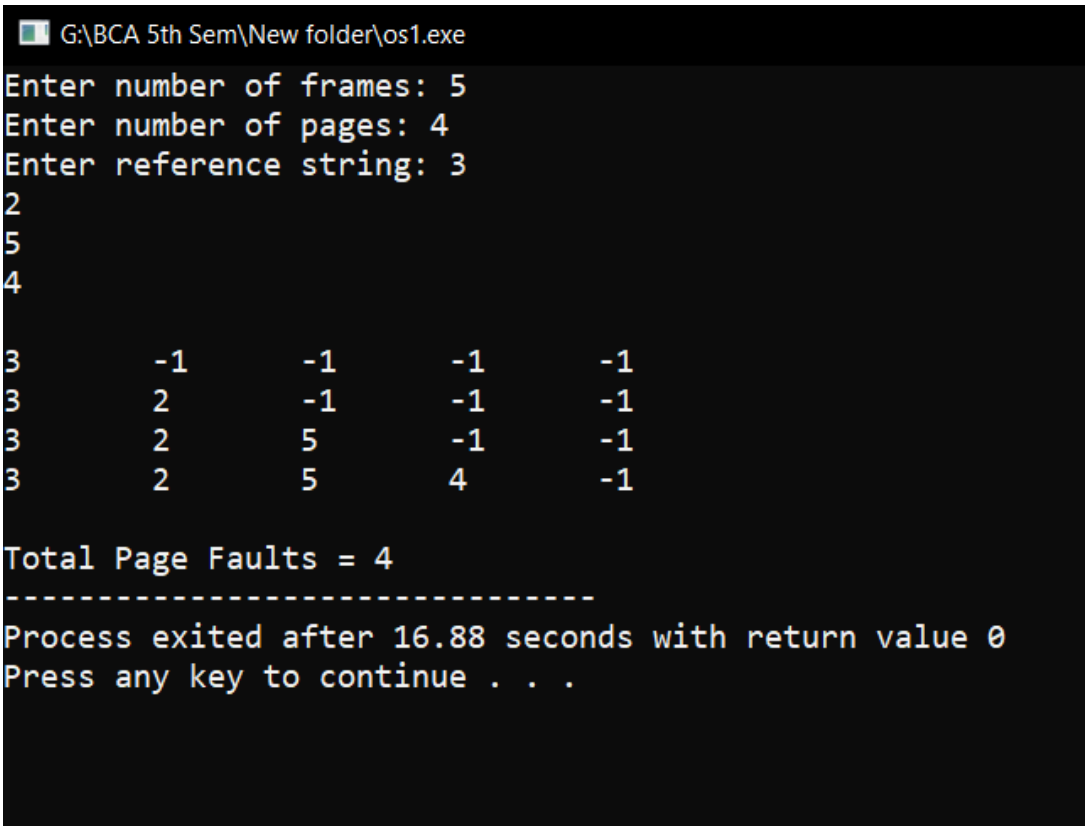
```
    return 0;
}
```


Select G:\BCA 5th Sem\New folder\os1.exe
Enter number of frames: 4
Enter number of pages: 3
Enter page reference string: 2
12
3
2 -1 -1 -1
2 12 -1 -1
2 12 3 -1
Total Page Faults = 3
Process exited after 13.54 seconds with return value 0
Press any key to continue . . .

LRU Page Replacement.

```
#include<stdio.h>

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;
}

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

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

   for(i = 0; i < no_of_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;
  }
```

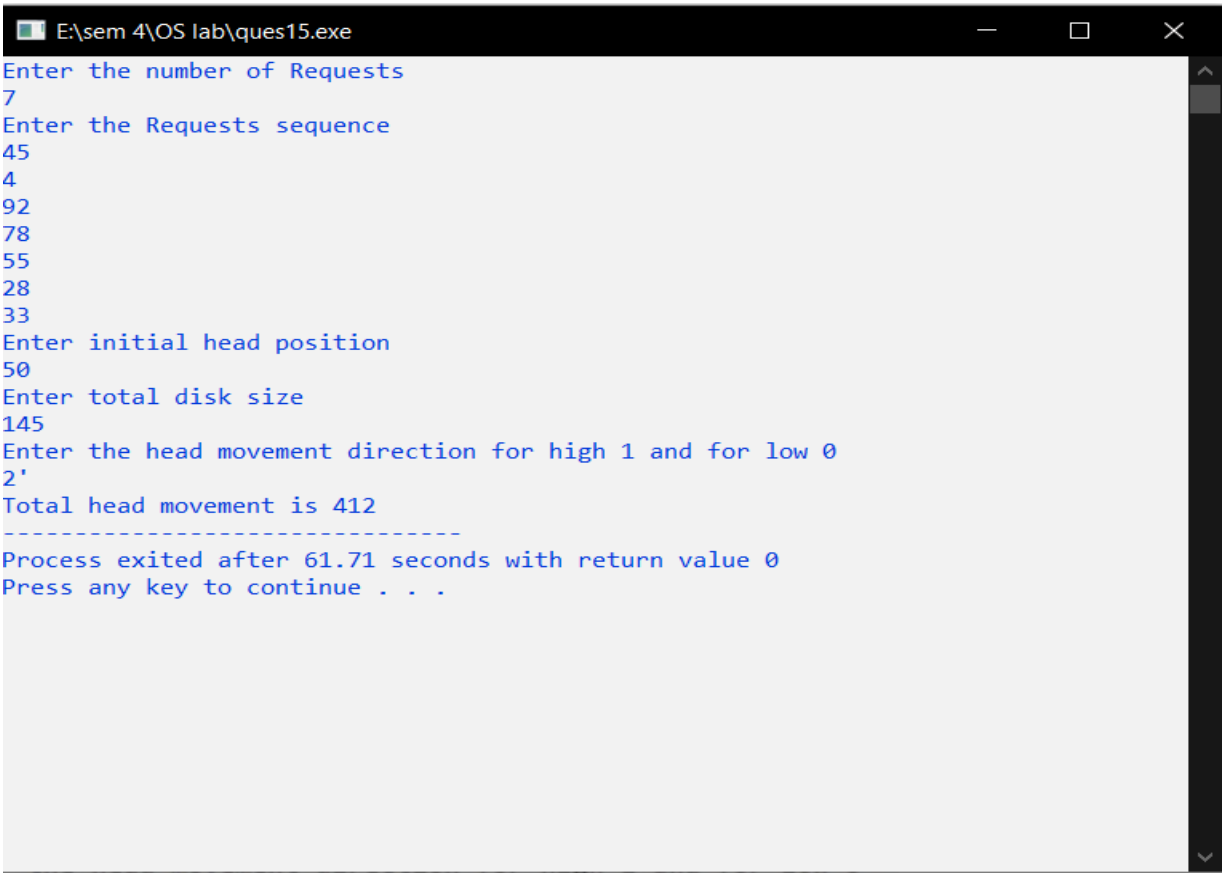
```
    }

    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;
    }

    printf("\n");

    for(j = 0; j < no_of_frames; ++j){
    printf("%d\t", frames[j]);
    }
}
printf("\n\nTotal Page Faults = %d", faults);

   return 0;
}
```


G:\BCA 5th Sem\New folder\os1.exe
Enter number of frames: 5
Enter number of pages: 4
Enter reference string: 3
2
5
4
3 -1 -1 -1 -1
3 2 -1 -1 -1
3 2 5 -1 -1
3 2 5 4 -1
Total Page Faults = 4
Process exited after 16.88 seconds with return value 0
Press any key to continue . . .

SCAN Disk Scheduling.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);
   printf("Enter total disk size\n");
   scanf("%d",&size);
   printf("Enter the head movement direction for high 1 and for low 0\n");
   scanf("%d",&move);

   // logic for Scan disk scheduling

      /*logic for sort the request array */
   for(i=0;i<n;i++)
   {
      for(j=0;j<n-i-1;j++)
      {
         if(RQ[j]>RQ[j+1])
         {
            int temp;
            temp=RQ[j];
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
         }

      }
   }

   int index;
   for(i=0;i<n;i++)
   {
      if(initial<RQ[i])
      {
         index=i;
         break;
```

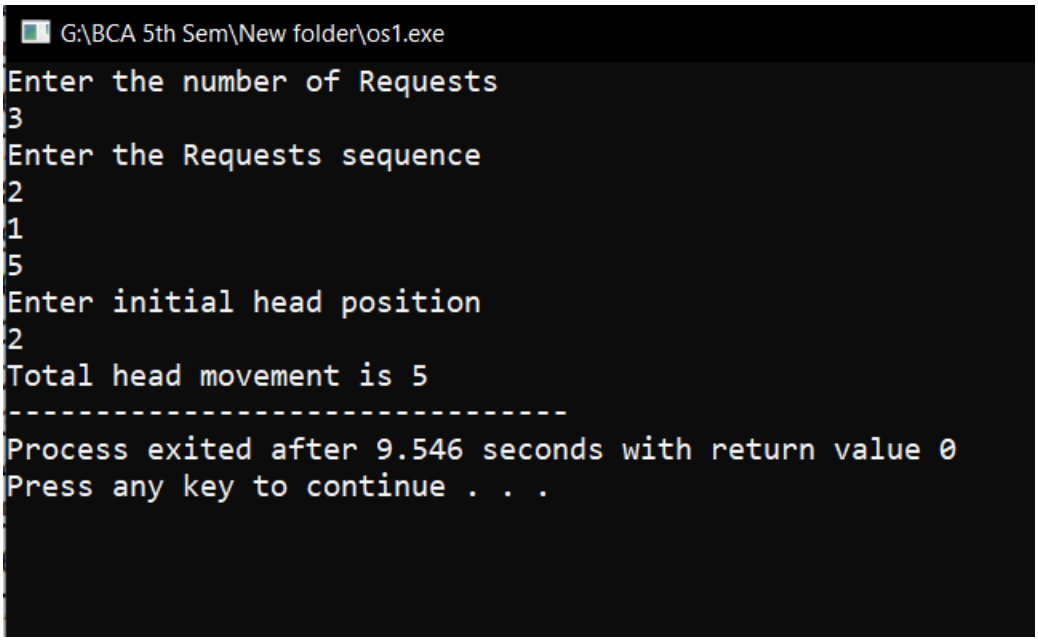
```
    }
  }
  // if movement is towards high value
  if(move==1)
  {
    for(i=index;i<n;i++)
    {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
    }
    // last movement for max size
    TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
    initial = size-1;
    for(i=index-1;i>=0;i--)
    {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];

    }
  }
  // if movement is towards low value else
  {
    for(i=index-1;i>=0;i--)
    {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
    }
    // last movement for min size
    TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
    initial =0;
    for(i=index;i<n;i++)
    {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];

    }
  }

  printf("Total head movement is %d",TotalHeadMoment);
  return 0;
}
```


E:\sem 4\OS lab\ques15.exe
Enter the number of Requests
7
Enter the Requests sequence
45
4
92
78
55
28
33
Enter initial head position
50
Enter total disk size
145
Enter the head movement direction for high 1 and for low 0
2'
Total head movement is 412

Process exited after 61.71 seconds with return value 0
Press any key to continue . . .

SSJF Disk Scheduling.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);

   // logic for sstf disk scheduling

      /* loop will execute until all process is completed*/
   while(count!=n)
   {
      int min=1000,d,index;
      for(i=0;i<n;i++)
      {
        d=abs(RQ[i]-initial);
        if(min>d)
        {
           min=d;
           index=i;
        }

      }
      TotalHeadMoment=TotalHeadMoment+min;
      initial=RQ[index];
      // 1000 is for max
      // you can use any number
      RQ[index]=1000;
      count++;
   }

   printf("Total head movement is %d",TotalHeadMoment);
   return 0;
}
```


G:\BCA 5th Sem\New folder\os1.exe
Enter the number of Requests
3
Enter the Requests sequence
2
1
5
Enter initial head position
2
Total head movement is 5

Process exited after 9.546 seconds with return value 0
Press any key to continue . . .