```
// ASS1 (Bankers Algorithm

#include<stdio.h>

            void main()
      {
            int k=0, output[10], d=0,t=0,ins[5], i,avail[5],allocated[10]
[5],need[10][5],MAX[10][5],pno,P[10],j,rz, count=0;


            printf("\n Enter the number of resources : ");
            scanf("%d", &rz);

            printf("\n enter the max instances of each resources\n");
            for (i=0;i<rz;i++) {
                  avail[i]=0;
                  printf("%c= ",(i+97));
                  scanf("%d",&ins[i]);
            }

            printf("\n Enter the number of processes : ");
            scanf("%d", &pno);

            printf("\n Enter the allocation matrix \n     ");
            for (i=0;i<rz;i++)
            printf(" %c",(i+97));
            printf("\n");
            for (i=0;i <pno;i++) {
            P[i]=i;
                  printf("P[%d]  ",P[i]);
            for (j=0;j<rz;j++) {
                        scanf("%d",&allocated[i][j]);
                        avail[j]+=allocated[i][j];
                  }
            }

   printf("\nEnter the MAX matrix \n     ");
            for (i=0;i<rz;i++) {
                  printf(" %c",(i+97));
                  avail[i]=ins[i]-avail[i];
            }
            printf("\n");
            for (i=0;i <pno;i++) {
                  printf("P[%d]  ",i);
                  for (j=0;j<rz;j++)
                   scanf("%d", &MAX[i][j]);
            }
            printf("\n");
            A: d=-1;
            for (i=0;i <pno;i++) {
                  count=0;
                  t=P[i];
                  for (j=0;j<rz;j++) {
                        need[t][j] = MAX[t][j]-allocated[t][j];
                        if(need[t][j]<=avail[j])
                         count++;
                  }
                  if(count==rz) {
                        output[k++]=P[i];
                        for (j=0;j<rz;j++)
                        avail[j]+=allocated[t][j];
                  } else
                   P[++d]=P[i];
            }
            if(d!=-1) {
                  pno=d+1;
                  goto A;
            }
            printf("\t <");
            for (i=0;i<k;i++)
            printf(" P[%d] ",output[i]);
            printf(">");

      }
```

```
// ASS2 (File Allocation Methods)

  Slot-I  :

    #include<stdio.h>
#include<conio.h>
main()
{
int n,i,j,b[20],sb[20],t[20],x,c[20][20];
clrscr();
printf("Enter no.of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter no. of blocks occupied by file%d",i+1);
scanf("%d",&b[i]);
printf("Enter the starting block of file%d",i+1);
scanf("%d",&sb[i]);
t[i]=sb[i];
for(j=0;j<b[i];j++)
c[i][j]=sb[i]++;
}
printf("Filename\tStart block\tlength\n");
for(i=0;i<n;i++)
printf("%d\t %d \t%d\n",i+1,t[i],b[i]);
printf("Enter file name:");
scanf("%d",&x);
printf("File name is:%d",x);
printf("length is:%d",b[x-1]);
printf("blocks occupied:");
for(i=0;i<b[x-1];i++)
printf("%4d",c[x-1][i]);
getch();
}
```

Slot-II :

```
      #include<stdio.h>
#include<conio.h>
struct file
{
char fname[10];
int start,size,block[10];
}f[10];
main()
{
int i,j,n;
clrscr();
printf("Enter no. of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter file name:");
scanf("%s",&f[i].fname);
printf("Enter starting block:");
scanf("%d",&f[i].start);
f[i].block[0]=f[i].start;
printf("Enter no.of blocks:");
scanf("%d",&f[i].size);
printf("Enter block numbers:");
for(j=1;j<=f[i].size;j++)
{
scanf("%d",&f[i].block[j]);
}
}
printf("File\tstart\tsize\tblock\n");for(i=0;i<n;i++)
{
printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
for(j=1;j<=f[i].size-1;j++)
printf("%d--->",f[i].block[j]);
printf("%d",f[i].block[j]);
printf("\n");
}
getch();
}
```

Slot-III :

```
      #include<stdio.h>
#include<conio.h>
main()
{
int n,m[20],i,j,sb[20],s[20],b[20][20],x;
clrscr();
printf("Enter no. of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter starting block and size of file%d:",i+1);
scanf("%d%d",&sb[i],&s[i]);
printf("Enter blocks occupied by file%d:",i+1);
scanf("%d",&m[i]);
printf("enter blocks of file%d:",i+1);
for(j=0;j<m[i];j++)
scanf("%d",&b[i][j]);
} printf("\nFile\t index\tlength\n");
for(i=0;i<n;i++)
{
printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);
}printf("\nEnter file name:");
scanf("%d",&x);
printf("file name is:%d\n",x);
i=x-1;
printf("Index is:%d",sb[i]);
printf("Block occupied are:");
for(j=0;j<m[i];j++)
printf("%3d",b[i][j]);
getch();
}
```

```
// ASS2 (Disk Scheduling Algorithm)


   Slot-I : i)

      #include<stdio.h>
#include<stdlib.h>
int main()
{
  int RQ[100],i,n,TotalHeadMove=0,initial;

  printf("Enter Number Of Requests : ");
  scanf("%d",&n);

  printf("Enter The Request Sequence : ");
  for(i=0;i<n;i++)
  scanf("%d",&RQ[i]);

  printf("Enter Initial Position of Head Read/Write : ");
  scanf("%d",&initial);

  for(i=0;i<n;i++)
  {
    TotalHeadMove=TotalHeadMove+abs(RQ[i]-initial);
    initial=RQ[i];
   }

   printf("Toatal Head Movment(Seek Time) = %d\n",TotalHeadMove);
   return 0;
}



    ii)
```

```
      # include<stdio.h>
# include<stdlib.h>
int main()
{
    int RQ[100],i,n,TotalHeadMove=0,initial,count=0,min,d,index;

    printf("enter number of Requests");
    scanf("%d",&n);

    printf("enter the Request Sequence");
    for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);

    printf("Enter initial position of head Read/Write");
    scanf("%d",&initial);

    while(count!=n)
    {
    min=1000;
    for(i=0;i<n;i++)
    {
        d=abs(RQ[i]-initial);
        if(min>d)
        {
        min=d;
        index=i;
        }
    }
    TotalHeadMove=TotalHeadMove+min;
    initial=RQ[index];
    RQ[index]=1000;
    count++;
    }

    printf("Total head Movement (Seek time) =%d",TotalHeadMove);
    return 0;


}
```

Slot-II : i)

```
         # include<stdio.h>
# include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMove=0,initial,count=0,min,d,index,move,size;

    printf("enter number of Requests");
    scanf("%d",&n);

    printf("enter the Request Sequence");
    for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
```

```
 printf("Enter initial position of head Read/Write");
 scanf("%d",&initial);

 printf("Enter initial Move");
 scanf("%d",&move);

 printf("Enter Size");
 scanf("%d",&size);

  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;
     }
     }
 }

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

 for(i=0;i<n;i++)
 {
   if(initial<RQ[i])
   {
     index=i;
     break;
  }
}
printf("\n index is : %d \n",index);

if(move==1)
{
    for(i=index;i<n;i++)
     {
        TotalHeadMove=TotalHeadMove+(abs(RQ[i]-initial));
        initial=RQ[i];
     }
TotalHeadMove=TotalHeadMove+abs(size-RQ[i-1]-1);
     initial = size-1;
     for(i=index-1;i>=0;i--)
     {
          TotalHeadMove=TotalHeadMove+abs(RQ[i]-initial);
          initial=RQ[i];

     }
 }
  else
 {
     for(i=index-1;i>=0;i--)
     {
            TotalHeadMove=TotalHeadMove+abs(RQ[i]-initial);
            initial=RQ[i];
        }
        TotalHeadMove=TotalHeadMove+abs(RQ[i+1]-0);
        initial =0;
        for(i=index;i<n;i++)
        {
             TotalHeadMove=TotalHeadMove+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }
   printf("\n TotalHeadMove is :  %d \n",TotalHeadMove);
   return 0;
}
```

ii)

```
#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);


    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(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
        TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
        TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
        initial=0;
        for( i=0;i<index;i++)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
        initial =size-1;
        for(i=n-1;i>=index;i--)
        {
             TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }

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

```
// ASS4 (Distrubuted And Mobile OS)

     Slot-I :

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

```c
#define n 10

int a2[1000];

int main(int argc, char* argv[])
{
  int a[1000],i;
  srand(time(NULL));
  for(i=0;i<1001;i++)
  {
        a[i]=(rand()%1000);
    }

    int pid, np,
        elements_per_process,
        n_elements_recieved;


    MPI_Status status;


    MPI_Init(&argc, &argv);


    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);


    if (pid == 0) {
        int index, i;
        elements_per_process = n / np;


        if (np > 1) {

            for (i = 1; i < np - 1; i++) {
                index = i * elements_per_process;

                MPI_Send(&elements_per_process,
                         1, MPI_INT, i, 0,
                         MPI_COMM_WORLD);
                MPI_Send(&a[index],
                         elements_per_process,
                         MPI_INT, i, 0,
                         MPI_COMM_WORLD);
            }


            index = i * elements_per_process;
            int elements_left = n - index;

            MPI_Send(&elements_left,
                     1, MPI_INT,
                     i, 0,
                     MPI_COMM_WORLD);
            MPI_Send(&a[index],
                     elements_left,
                     MPI_INT, i, 0,
                    MPI_COMM_WORLD);
        }


        int sum = 0;
        for (i = 0; i < elements_per_process; i++)
            sum += a[i];


        int tmp;
        for (i = 1; i < np; i++) {
            MPI_Recv(&tmp, 1, MPI_INT,
                     MPI_ANY_SOURCE, 0,
                     MPI_COMM_WORLD,
                     &status);
            int sender = status.MPI_SOURCE;

            sum += tmp;
        }


        printf("Sum of array is : %d\n", sum);
    }

    else {
        MPI_Recv(&n_elements_recieved,
                 1, MPI_INT, 0, 0,
                 MPI_COMM_WORLD,
                 &status);

        MPI_Recv(&a2, n_elements_recieved,
                 MPI_INT, 0, 0,
                 MPI_COMM_WORLD,
                 &status);


        int partial_sum = 0,i;
        for (i = 0; i < n_elements_recieved; i++)
            partial_sum += a2[i];


        MPI_Send(&partial_sum, 1, MPI_INT,
                 0, 0, MPI_COMM_WORLD);
    }

    MPI_Finalize();

    return 0;
}
```

Slot-II :

```
#include "mpi.h"
```

```
/* MPI header file */
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#define MAX_LEN 100000 /* Max array size */
/* Usual search for largest function */
int find_max(int a[ ],int len)
{
int i;
int max; /* Current max */
max = a[0];
for (i=1;i<len;++i)
if (a[i] > max) max = a[i];
return max;}
/* Function to generate random ints */
void generate_data (int a[ ],int len)
{
int i;
struct timeval time;
/* Use time of day to get a seed */
gettimeofday(&time, (struct timezone *) 0);
srand((int) time.tv_sec);
for (i=0;i<len;++i)
a[i] = rand();
}
int main (int argc, char *argv[])
{
int my_rank;
int rank; /* Loop variable for the processes */
int num_proc;
/* Total number of processes */
int array_len; /* Length of the main array */
int sequential; /* Should we do sequential */
int quotient; /* Usual subarray size: array_len/num_proc*/
int rem; /* How many larger subarrays:
array_len % num_proc */
int sub_start; /* Start of one of the subarrays */
int sub_len; /* Length of my subarray */int search_array[MAX_LEN]; /* The array to
search */
int my_max; /* Max for my subarray */
int global_max; /* Maximum for the main array */
int local_max; /* Local max from one process */
MPI_Status status; /* status for receive */
/* Usual startup tasks */
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
/* Code for Process 0 */
if (my_rank == 0) {
sequential = atoi(argv[1]); /* Sequential 1 - Parallel !=1 */
array_len = atoi(argv[2]); /* Array dimension on compute the max*/
MPI_Comm_size(MPI_COMM_WORLD, &num_proc);
/* Get values needed for subarray sizes */
quotient = array_len / num_proc; /*P part of N/P elements*/
rem = array_len % num_proc; /*Number of processes that need an additional element*/
generate_data(search_array,array_len);
if (sequential)
printf("The sequential search gives %d\n",
find_max(search_array,array_len));
/* Some subarrays may be larger */for (rank=1; rank < rem; ++rank){
```

```
sub_len = quotient+1;
/*rank * quotient, is the number of element in the
part before your part
+ rank how many part of size 1 is before
you?*/
sub_start = rank*quotient+rank;
MPI_Send(&sub_len,1,MPI_INT,rank,0,MPI_COMM_WORLD);
MPI_Send(&(search_array[sub_start]),sub_len, MPI_INT,
rank, 0, MPI_COMM_WORLD);
}
for (rank=rem; rank < num_proc; ++rank){
sub_len = quotient;
/*rank * quotient, is the number of element in the
part before your part
+ rem how many part of size 1 is before
you?*/
sub_start = rank*quotient+rem;
/* Send the process its subarray length */
MPI_Send(&sub_len,1,MPI_INT,rank,0,MPI_COMM_WORLD);
/* And send the subarray */
MPI_Send(&(search_array[rank*quotient+rem]),quotient,
MPI_INT, rank, 0, MPI_COMM_WORLD);
}
/* Find my local max */
if (rem==0)
sub_len=quotient;
else sub_len=quotient+1;global_max = find_max(search_array,quotient+1);
/* Get back the maxima from the others */
for (rank=1;rank<num_proc;++rank){
MPI_Recv(&local_max,1,MPI_INT,MPI_ANY_SOURCE,0,
MPI_COMM_WORLD, &status);
if (local_max > global_max)
global_max = local_max;
}
/* Display the global max */
printf("The parallel search gives %d\n", global_max);
}
else { /* Code for other processes */
/* Receive my subarray length */
MPI_Recv(&sub_len,1,MPI_INT,0,0,MPI_COMM_WORLD,&status);
/* And receive the subarray */
MPI_Recv(search_array,sub_len,MPI_INT,0,0,MPI_COMM_WORLD,
&status);
my_max = find_max(search_array,sub_len);
/* Send back my local max */
MPI_Send(&my_max,1,MPI_INT,0,0,MPI_COMM_WORLD);
}
MPI_Finalize();
return 0;
}
/* And close up MPI */
```