```c
// 6. a) Write a C program to implement Banker's Algorithm(Safety Algorithm)
#include <stdio.h>
int main()
{
    int m,n,i,j,k,y,alloc[20][20],max[20][20],avail[50],ind=0;
    printf("Enter the no. of process and resources:");
    scanf("%d %d",&n,&m);
    printf("Enter the allocation matrix:\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        scanf("%d",&alloc[i][j]);
    }
    printf("Enter the max matrix:\n");
     for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        scanf("%d",&max[i][j]);
    }
    printf("Enter available matrix\n");
    for(i=0;i<m;i++)
    scanf("%d",&avail[i]);
    int finish[n],safeseq[n],work[m],need[n][m];
    for(i=0;i<m;i++)
    {
        work[i]=avail[i];
    }
    for(i=0;i<n;i++)
    {
        finish[i]=0;
    }
     for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        need[i][j]=max[i][j]-alloc[i][j];
    }
    printf("Need matrix is");
    for(i=0;i<n;i++)
    {
        printf("\n");
        for(j=0;j<m;j++)
        printf("%d ",need[i][j]);
    }
    for(i=0;i<m;i++)
    {
        work[i]=avail[i];
    }
    for(i=0;i<n;i++)
    {
        finish[i]=0;
    }
    for(k=0;k<n;k++)
    {
        for(i=0;i<n;i++)
        {
            if(finish[i]==0)
            {
                int flag=0;
```

Output

```
Enter the no. of process and resources:5
4
Enter the allocation matrix:
0 0 1 2
1 0 0 0
1 3 5 4
0 6 3 2
0 0 1 4
Enter the max matrix:
0 0 1 2
1 7 5 0
2 3 5 6
0 6 5 2
0 6 5 6
Enter available matrix
1 5 2 0
Need matrix is
0 0 0 0
0 7 5 0
1 0 0 2
0 0 2 0
0 6 4 2
Following is the safe seq
P0 P2 P3 P4 P1
```

```
                for(j=0;j<m;j++)
                {
                    if(need[i][j]>work[j])
                    {
                        flag=1; break;
                    }
                }
                if(flag==0)
                {
                    safeseq[ind++]=i;
                    for(y=0;y<m;y++)
                    work[y]+=alloc[i][y];
                    finish[i]=1;
                }
            }
        }
    }
    printf("\nFollowing is the safe seq\n");
    for(i=0;i<=n-1;i++)
    printf("P%d ",safeseq[i]);
}
```

```c
//7.a)Write a C Program to implement the First fit Algorithm.
#include <stdio.h>
void firstFit(int blocks[], int m, int process[], int n)
{
    int allocation[10];
    for (int i = 0; i < n; i++)
        allocation[i] = -1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (blocks[j] >= process[i])
            {
                allocation[i] = j;

                blocks[j] -= process[i];

                break;
            }
        }
    }

    printf("Process No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d\t\t%d\t\t", i + 1, process[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

int main()
{
    int blocks[10];
    int process[10];
    int m, n;

    printf("Enter the number of memory blocks: ");
    scanf("%d", &m);

    printf("Enter the size of each memory block:\n");
    for (int i = 0; i < m; i++)
    {
        scanf("%d", &blocks[i]);
    }

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

    printf("Enter the size of each process:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &process[i]);
    }

    firstFit(blocks, m, process, n);

    return 0;
}
```

Output

```
Enter the number of memory blocks: 5
Enter the size of each memory block:
100
500
200
300
600
Enter the number of processes: 4
Enter the size of each process:
212
417
112
426
Process No.     Process Size    Block No.
1               212             2
2               417             5
3               112             2
4               426             Not Allocated
```

```c
//7.b)Write a C Program to implement the Best fit Algorithm.
#include <stdio.h>
void BestFit(int blocks[], int m, int process[], int n)
{
    int allocation[10];
    for (int i = 0; i < n; i++)
        allocation[i] = -1;
     for (int i = 0; i < n; i++)
    {
        int bestFit = -1;

        for (int j = 0; j < m; j++)
        {
            if (blocks[j] >= process[i])
            {
                if (bestFit == -1 || blocks[j] < blocks[bestFit])
                    bestFit = j;
            }
        }
        if (bestFit!= -1)
        {
            allocation[i] = bestFit;
            blocks[bestFit] -= process[i];
        }
    }
    printf("Process No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d\t\t%d\t\t", i + 1, process[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

int main()
{
    int blocks[10];
    int process[10];
    int m, n;

    printf("Enter the number of memory blocks: ");
    scanf("%d", &m);

    printf("Enter the size of each memory block:\n");
    for (int i = 0; i < m; i++)
    {
        scanf("%d", &blocks[i]);
    }
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter the size of each process:\n");
    for (int i = 0; i < n; i++)
    {
        printf("Enter the size of process %d: ",i+1);
        scanf("%d", &process[i]);
    }

    BestFit(blocks, m, process, n);

    return 0;
}
```

Output

```
Enter the number of memory blocks: 5
Enter the size of each memory block:
100
500
200
300
600
Enter the number of processes: 4
Enter the size of each process:
Enter the size of process 1: 212
Enter the size of process 2: 417
Enter the size of process 3: 112
Enter the size of process 4: 426
Process No.     Process Size    Block No.
1               212             4
2               417             2
3               112             3
4               426             5
```

```c
//7.c)Write a C Program to implement the Worst fit Algorithm.
#include <stdio.h>
void BestFit(int blocks[], int m, int process[], int n)
{
    int allocation[10];
    for (int i = 0; i < n; i++)
        allocation[i] = -1;
     for (int i = 0; i < n; i++)
    {
        int bestFit = -1;

        for (int j = 0; j < m; j++)
        {
            if (blocks[j] >= process[i])
            {
                if (bestFit == -1 || blocks[j] > blocks[bestFit])
                    bestFit = j;
            }
        }
        if (bestFit!= -1)
        {
            allocation[i] = bestFit;
            blocks[bestFit] -= process[i];
        }
    }
    printf("Process No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d\t\t%d\t\t", i + 1, process[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

int main()
{
    int blocks[10];
    int process[10];
    int m, n;

    printf("Enter the number of memory blocks: ");
    scanf("%d", &m);

    printf("Enter the size of each memory block:\n");
    for (int i = 0; i < m; i++)
    {
        scanf("%d", &blocks[i]);
    }
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    printf("Enter the size of each process:\n");
    for (int i = 0; i < n; i++)
    {
        printf("Enter the size of process %d: ",i+1);
        scanf("%d", &process[i]);
    }
     BestFit(blocks, m, process, n);
     return 0;
}
```

Output

```
Enter the number of memory blocks: 5
Enter the size of each memory block:
100
500
200
300
600
Enter the number of processes: 4
Enter the size of each process:
Enter the size of process 1: 212
Enter the size of process 2: 417
Enter the size of process 3: 112
Enter the size of process 4: 426
Process No.     Process Size    Block No.
1               212             5
2               417             2
3               112             5
4               426             Not Allocated
```

```c
//8a PAGE REPLACEMENT ALGORITHM - FIFO
#include<stdio.h>
void main() {
 int i, j, k, f, pf=0,ph=0, count=0, rs[25], m[10], n;
 printf("\n Enter the length of reference string -- ");
 scanf("%d",&n); printf("\n Enter the reference string -- ");
 for(i=0;i<n;i++) scanf("%d",&rs[i]);
 printf("\n Enter no. of frames -- ");
 scanf("%d",&f);
 for(i=0;i<f;i++) m[i]=-1;
 printf("\n The Page Replacement Process is -- \n");

 for(i=0;i<n;i++)
    {
        for(k=0;k<f;k++)
        {
            if(m[k]==rs[i])
            break;
        }
        if(k==f)
        {
            m[count++]=rs[i];
            pf++;
        }
        if(k!=f)
        {
            ph++;
        }
        for(j=0;j<f;j++)
        printf("\t%d",m[j]);
        if(k!=f) printf("\tPh No. %d",ph);
        if(k==f)
        printf("\tPF No. %d",pf);
        printf("\n");
        if(count==f) count=0;
    }
 printf("\n The number of Page Faults using FIFO are %d",pf);
 printf("\n The number of Page Hits using FIFO are %d",n-pf);
}
```

Output

```
Enter the length of reference string -- 12
Enter the reference string -- 2 3 2 1 5 2 4 5 3 2 5 2
Enter no. of frames -- 3
The Page Replacement Process is --
        2       -1      -1      PF No. 1
        2       3       -1      PF No. 2
        2       3       -1      Ph No. 1
        2       3       1       PF No. 3
        5       3       1       PF No. 4
        5       2       1       PF No. 5
        5       2       4       PF No. 6
        5       2       4       Ph No. 2
        3       2       4       PF No. 7
        3       2       4       Ph No. 3
        3       5       4       PF No. 8
        3       5       2       PF No. 9
The number of Page Faults using FIFO are 9
The number of Page Hits using FIFO are 3
```

```c
//8b LRU PROGRAM
#include<stdio.h>
void main()
{
    int i,j,k,min,rs[25],m[10],count[10],flag[25],n,f,pf=0,next=0;
    printf("Enter the length of the reference string --");
    scanf("%d",&n);
    printf("Enter the reference string -- ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&rs[i]);
        flag[i]=0;
    }
    printf("Enter the number of frames -- ");
    scanf("%d",&f);
    for(i=0;i<f;i++)
    {
        count[i]=0;
        m[i]=-1;
    }
printf("\nThe Page Replacement process is -- \n");
for(i=0;i<n;i++)
{
    for(j=0;j<f;j++)
    {
        if(m[j]==rs[i])
        {
            flag[i]=1;
            count[j]=next;
            next++;
        }
    }
    if(flag[i]==0)
    {
        if(i<f)
        {
            m[i]=rs[i];
            count[i]=next;
            next++;
        }
        else
        {
            min=0;
            for(j=0;j<f;j++)
            if(count[min] > count[j])
                min=j;
            m[min]=rs[i];
            count[min]=next;
            next++;
        }
        pf++;
    }
    for(j=0;j<f;j++)
    printf("%d\t",m[j]);
    if(flag[i]==0)
    printf("PF no. -- %d",pf);
    printf("\n");
}
printf("\nThe number of page faults using LRU are %d",pf);
printf("\nThe number of page hits using LRU are %d",n-pf);
}
```

Output

```
Enter the length of the reference string --12
Enter the reference string -- 2 3 2 1 5 2 4 5 3 2 5 2
Enter the number of frames -- 3
The Page Replacement process is --
2       -1      -1      PF no. -- 1
2       3       -1      PF no. -- 2
2       3       -1
2       3       1       PF no. -- 3
2       5       1       PF no. -- 4
2       5       1
2       5       4       PF no. -- 5
2       5       4
3       5       4       PF no. -- 6
3       5       2       PF no. -- 7
3       5       2
3       5       2
The number of page faults using LRU are 7
The number of page hits using LRU are 5
```

```c
//8c PAGE REPLACEMENT ALGORITHM-OPTIMAL
#include <stdio.h>
int main() {
    int i, j, k, rs[25], m[10], flag[25] = {0}, n, f, pf = 0;

    printf("Enter the length of the reference string: ");
    scanf("%d", &n);
    printf("Enter the reference string: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &rs[i]);
    }
    printf("Enter the number of frames: ");
    scanf("%d", &f);

    printf("\nThe Page Replacement process is --\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < f; j++) {
            if (m[j] == rs[i]) {
                flag[i] = 1;
                break;
            }
        }
        if (flag[i] == 0) {
            int replaceIndex = -1;
            int farthestIndex = i + 1;
            for (j = 0; j < f; j++) {
                int pageFoundLater = 0;
                for (k = i + 1; k < n; k++) {
                    if (m[j] == rs[k]) {
                        pageFoundLater = 1;
                        if (k > farthestIndex) {
                            farthestIndex = k;
                            replaceIndex = j;
                        }
                        break;
                    }
                }
                if (!pageFoundLater) {
                    replaceIndex = j;
                    break;
                }
            }
            m[replaceIndex] = rs[i];
            pf++;
        }
        for (j = 0; j < f; j++) {
            printf("%d\t", m[j]);
        }
        if (flag[i] == 0) {
            printf("PF no. -- %d", pf);
        }
        printf("\n");
    }
    printf("\nThe number of page faults using Optimal algorithm: %d\n", pf);
 printf("\nThe number of page hits using Optimal algorithm: %d\n",n-pf);
 return 0;
}
```

Output

```
Enter the length of the reference string: 12
Enter the reference string: 2 3 2 1 5 2 4 5 3 2 5 2
Enter the number of frames: 3
The Page Replacement process is --
2       0       0       PF no. -- 1
2       3       0       PF no. -- 2
2       3       0
2       3       1       PF no. -- 3
2       3       5       PF no. -- 4
2       3       5
4       3       5       PF no. -- 5
4       3       5
4       3       5
2       3       5       PF no. -- 6
2       3       5
2       3       5
The number of page faults using Optimal algorithm: 6
The number of page hits using Optimal algorithm: 6
```

```c
//9a_FCFS Program to simulate the FCFS disk scheduling algorithm
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n,req[50],mov=0,cp;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
    scanf("%d",&n);
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&req[i]);
    }
    mov=mov+abs(cp-req[0]); // abs is used to calculate the absolute value
    printf("%d -> %d",cp,req[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(req[i]-req[i-1]);
        printf(" -> %d",req[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

Output

```
enter the current position
5
enter the number of requests
6
enter the request order
2 6 4 1 5 4
5 -> 2 -> 6 -> 4 -> 1 -> 5 -> 4
total head movement = 17
```

```c
//9b_SSTF.Program to implement the SSTF disk scheduling algorithm.
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,n,k,req[50],mov=0,cp,index[50],min,a[50],j=0,mini,cp1;
    printf("enter the current position\n");
    scanf("%d",&cp);
    printf("enter the number of requests\n");
    scanf("%d",&n);
    cp1=cp;
    printf("enter the request order\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&req[i]);
    }
    for(k=0;k<n;k++)
    {
    for(i=0;i<n;i++)
    {
        index[i]=abs(cp-req[i]); //calculate distance of each
            request from current position
    }
    // to find the nearest request
    min=index[0];
    mini=0;
    for(i=1;i<n;i++)
    {
        if(min>index[i])
        {
            min=index[i];
            mini=i;
        }
    }
    a[j]=req[mini];
```

Output

```
enter the current position
50
enter the number of requests
5
enter the request order
34
87
45
77
22
Sequence is : 50 -> 45 -> 34 -> 22 -> 77 -> 87
total head movement = 93
```

```
    j++;
    cp=req[mini]; // change the current position value to next request
    req[mini]=999;
    } // the request that is processed its value is changed so that it is not
        processed again
    printf("Sequence is : ");
    printf("%d",cp1);
    mov=mov+abs(cp1-a[0]);    // head movement
    printf(" -> %d",a[0]);
    for(i=1;i<n;i++)
    {
        mov=mov+abs(a[i]-a[i-1]); ///head movement
        printf(" -> %d",a[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

```
//9c_SCAN.Write a C Program to implement the SCAN disk scheduling algorithm.
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int RQ[100],i,in,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the  no. of requests\n");
    scanf("%d",&n);
    printf("Enter the requests seq:");
    for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
    printf("Enter initial head position:");
    scanf("%d",&initial);
    in=initial;
    printf("Enter total disk size:");
    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);
```

Output

```
Enter the  no. of requests:8
Enter the requests seq:98
183
37
122
14
124
65
67
Enter initial head position:53
Enter total disk size:200
Enter the head movement direction for high 1 and for low 0
0
Total HM is 236
Sequence of Head Movements: 53 37 14 0 65 67 98 122 124 183
```

```
            initial=RQ[i];
        }
        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];
        }
    }
    else
    {
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
        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 HM is %d",TotalHeadMoment);
    printf("\nSequence of Head Movements: ");
    printf("%d ", in); // Print initial head position
    if (move == 1) {
        for (i = index; i < n; i++) {
            printf("%d ", RQ[i]);
        }
        printf("%d ", size - 1); // Print the end of the disk
        for (i = index - 1; i >= 0; i--) {
            printf("%d ", RQ[i]);
        }
    } else {
        for (i = index - 1; i >= 0; i--) {
            printf("%d ", RQ[i]);
        }
        printf("0 "); // Print the beginning of the disk
        for (i = index; i < n; i++) {
            printf("%d ", RQ[i]);
        }
    }
printf("\n");
return 0;
}
```

```c
//10_a Program to simulate the sequential file allocation technique
#include <stdio.h>
int main() {
    int n, i, j, b[20], sb[20], t[20], x, c[20][20];
    printf("Enter the number of files: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("Enter the number 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("\nFilename\tStart block\tLength\n");
    for (i = 0; i < n; i++)
        printf("%d\t\t%d\t\t%d\n", i + 1, t[i], b[i]);
    printf("\nEnter the file name: ");
    scanf("%d", &x);
    printf("File name is: %d\n", x);
    printf("Length is: %d\n", b[x - 1]);
    printf("Blocks occupied: ");
    for (i = 0; i < b[x - 1]; i++)
        printf("%4d", c[x - 1][i]);
    return 0;
}
```

Output

```
Enter the number of files: 2
Enter the number of blocks occupied by file 1: 4
Enter the starting block of file 1: 5
Enter the number of blocks occupied by file 2: 3
Enter the starting block of file 2: 3
Filename        Start block     Length
1               5               4
2               3               3
Enter the file name: 1
File name is: 1
Length is: 4
Blocks occupied:    5   6   7   8
```

```c
//10_b.Program to implement the linked file allocation technique
#include<stdio.h>
struct file
{
    char fname[10];
    int start, size, block[10];
};
int main()
{
    struct file f[10];
    int i, j, n;
    printf("Enter the number 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 the number 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");
    }
    return 0;
}
```

Output

```
Enter the number of files: 2
Enter file name: 1
Enter starting block: 5
Enter the number of blocks: 3
Enter block numbers: 4
1
2
Enter file name: 2
Enter starting block: 4
Enter the number of blocks: 6
Enter block numbers: 7
3
5
9
8
6
File    start   size    block
1       5       3       4--->1--->2
2       4       6       7--->3--->5--->9--->8--->6
```

```c
//10_c.Program to implement the indexed file allocation technique
#include<stdio.h>
int main()
{
    int n, m[20], i, j, sb[20], s[20], b[20][20], x;
    printf("Enter the number 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\tindex\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\n", sb[i]);
    printf("Blocks occupied are: ");
    for(j = 0; j < m[i]; j++)
        printf("%3d", b[i][j]);
    return 0;
}
```

Output

```
Enter the number of files: 2
Enter starting block and size of file 1: 2 10
Enter blocks occupied by file 1: 3
Enter blocks of file 1: 4 5 3
Enter starting block and size of file 2: 4
10
Enter blocks occupied by file 2: 3
Enter blocks of file 2: 6 7 8
File    index   length
1       2       3
2       4       3
Enter file name: 1
File name is: 1
Index is: 2
Blocks occupied are:   4  5  3
```