PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 5.1*/

#include<stdio.h>
#include<stdlib.h>
void ff(int mem[],int mc,int pro[],int pc)
{
    int occupied[20],allocated[20],i,frag=0;
    for(i =0;i<mc;i++)
    {
        occupied[i] = 0;
    }
    for(i=0;i<pc;i++)
    {
        allocated[i] = -1;
    }
    for(i=0;i<pc;i++)
    {
        for(int j=0;j<mc;j++)
        {
            if((pro[i]<= mem[j]) && (occupied[j] == 0))
            {
                allocated[i]=j;
                occupied[j]=1;
                frag += mem[j] - pro[i];
                break;
            }
        }
    }
    printf("\nFIRST FIT ALLOCATION\n");
    printf("Process no\tProcess size\tBlock no\n");
    for(i=0;i<pc;i++)
    {
        if(allocated[i]!= -1)
        {
            printf("%d\t\t%d\t\t%d\n",i+1,pro[i],allocated[i]+1);
        }
        else
        {
            printf("%d\t\t%d\t\tNot allocated\n",i+1,pro[i]);
        }
    }
     printf("Total Internal Fragmentation: %d\n", frag);
}
void bf(int mem[],int mc,int pro[],int pc)
{
    int occupied[20],allocated[20],i,frag=0;
    for(i =0;i<mc;i++)
    {
        occupied[i] = 0;
    }
    for(i=0;i<pc;i++)
    {
        allocated[i] = -1;
    }
    for(i=0;i<pc;i++)
    {
        int bindex=-1;
        for(int j=0;j<mc;j++)
        {
            if((pro[i]<= mem[j]) && (occupied[j] == 0))
            {
                if(bindex==-1 || mem[j]<mem[bindex])
                {
                    bindex=j;
                }
            }
        }
        if(bindex!=-1)
        {
            allocated[i]=bindex;
            occupied[bindex]=1;
            frag += mem[bindex] - pro[i];
        }
    }
    printf("\nBEST FIT ALLOCATION\n");
    printf("Process no\tProcess size\tBlock no\n");
    for(i=0;i<pc;i++)
    {
        if(allocated[i]!= -1)
        {
            printf("%d\t\t%d\t\t%d\n",i+1,pro[i],allocated[i]+1);
        }
        else
        {
            printf("%d\t\t%d\t\tNot allocated\n",i+1,pro[i]);
        }
    }
     printf("Total Internal Fragmentation: %d\n", frag);
}
void wf(int mem[],int mc,int pro[],int pc)
{
    int occupied[20],allocated[20],i,frag=0;
    for(i =0;i<mc;i++)
    {
        occupied[i] = 0;
    }
    for(i=0;i<pc;i++)
    {
        allocated[i] = -1;
    }
    for(i=0;i<pc;i++)
    {
        int windex=-1;
        for(int j=0;j<mc;j++)
        {
            if((pro[i]<= mem[j]) && (occupied[j] == 0))
            {
                if(windex==-1 || mem[j]>mem[windex])
                {
                    windex=j;
                }
            }
        }
        if(windex!=-1)
        {
            allocated[i]=windex;
            occupied[windex]=1;
            frag += mem[windex] - pro[i];
        }
    }
    printf("\nWORST FIT ALLOCATION\n");
    printf("Process no\tProcess size\tBlock no\n");
    for(i=0;i<pc;i++)
    {
        if(allocated[i]!= -1)
        {
            printf("%d\t\t%d\t\t%d\n",i+1,pro[i],allocated[i]+1);
        }
        else
        {
            printf("%d\t\t%d\t\tNot allocated\n",i+1,pro[i]);
        }
    }
     printf("Total Internal Fragmentation: %d\n", frag);
}
int main()
{
    int mc,pc,mem[20],pro[20];
    printf("Enter the number of memory blocks\n");
    scanf("%d",&mc);
    for(int i=0;i<mc;i++)
    {
        printf("Enter size of memory block %d\n",i+1);
        scanf("%d",&mem[i]);
    }
    printf("Enter the number of processes\n");
    scanf("%d",&pc);
    for(int i=0;i<pc;i++)
    {
        printf("Enter size of process %d\n",i+1);
        scanf("%d",&pro[i]);
    }
    int choice;
    while(1)
    {
      printf("1.First fit\n2.Best fit\n3.Worst fit\n4.Exit\n");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
             ff(mem,mc,pro,pc);
             break;
         case 2:
             bf(mem,mc,pro,pc);
             break;
         case 3:
             wf(mem,mc,pro,pc);
             break;
         case 4:
             exit(0);
      }
    }
}
```

OUTPUT

```
Enter the number of memory blocks
5
Enter size of memory block 1
100
Enter size of memory block 2
500
Enter size of memory block 3
200
Enter size of memory block 4
300
Enter size of memory block 5
600
Enter the number of processes
4
Enter size of process 1
212
Enter size of process 2
417
Enter size of process 3
112
Enter size of process 4
426
1.First fit
2.Best fit
3.Worst fit
4.Exit
1

FIRST FIT ALLOCATION
```

```
Process no    Process size    Block no
1         212         2
2         417         5
3         112         3
4         426         Not allocated
Total Internal Fragmentation: 559
1.First fit
2.Best fit
3.Worst fit
4.Exit
2

BEST FIT ALLOCATION
Process no    Process size    Block no
1         212         4
2         417         2
3         112         3
4         426         5
Total Internal Fragmentation: 433
1.First fit
2.Best fit
3.Worst fit
4.Exit
3

WORST FIT ALLOCATION
Process no    Process size    Block no
1         212         5
2         417         2
3         112         4
4         426         Not allocated
Total Internal Fragmentation: 659
1.First fit
2.Best fit
3.Worst fit
4.Exit
4
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 2.1*/

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>
#include<stdbool.h>
#include<math.h>
bool isprime(int i)
{
    if(i<2)
        return false;
    else
    {
        for(int j=2;j<=sqrt(i);j++)
        {
            if(i%j==0)
            {
                return false;
            }
        }
    }
    return true;
}
void prime(int n)
{
    int count=0,i=2;
    printf("\nThe first %d prime numbers are :",n);
    while(count<n)
    {
        if(isprime(i))
        {
            printf("%d\t",i);
            count++;
        }
        i++;
    }
}
void fibonacci(int n)
{
    int a,b,c;
    a=0;
    b=1;
    printf("\nFibonacci series: %d\t%d\t",a,b);
    for(int i=2;i<n;i++)
    {
        c=a+b;
        printf("%d\t",c);
        a=b;
        b=c;
    }
    printf("\n");
}
int main()
{
    int n;
    printf("Enter the limiting value\n");
    scanf("%d",&n);
    pid_t pid = fork();
    if(pid < 0)
    {
        printf("error\n");
        exit(0);
    }
    else if(pid == 0)
    {
        fibonacci(n);
    }
    else
    {
        prime(n);
        wait(NULL);
    }
    return 0;
}
```

OUTPUT

```
Enter the limiting value
7

Fibonacci series: 0     1       1       2       3       5       8
The first 7 prime numbers are :2        3       5       7       11      13      17
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 2.2*/

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/wait.h>

int main()
{
    pid_t pid;
    printf("Process A : ID - %d\t Parent ID - %d\n",getpid(),getppid());
    if((pid = fork()) == -1)////////////////////////////B
    {
        perror("fork failed");
        exit(1);
    }
    if(pid == 0)
    {
        printf("Process B : ID - %d\t Parent ID - %d\n",getpid(),getppid());
        if((pid = fork()) == -1)////////////////////////////D
        {
            perror("fork failed");
            exit(1);
        }
        if(pid == 0)
        {
            printf("Process D : ID - %d\t Parent ID - %d\n",getpid(),getppid());
            if((pid = fork()) == -1)////////////////////////////H
            {
                perror("fork failed");
                exit(1);
            }
            if(pid == 0)
            {
                printf("Process H : ID - %d\t Parent ID - %d\n",getpid(),getppid());
                if((pid = fork()) == -1)////////////////////////////I
                {
                    perror("fork failed");
                    exit(1);
                }
                if(pid == 0)
                {
                printf("Process I : ID - %d\t Parent ID - %d\n",getpid(),getppid());
                }
            }
        }
        else if((pid = fork()) == -1)////////////////////////////E
        {
         perror("fork failed");
         exit(1);
      }
      else if(pid == 0)
      {
         printf("Process E : ID - %d\t Parent ID - %d\n",getpid(),getppid());
      }
      else
      {
         pid= fork();///////////F
         if(pid==-1)
         {
            perror("failed");
         }
         else if(pid==0)
         {
            printf("Process F : ID - %d\t Parent ID - %d\n",getpid(),getppid());
         }
      }
   }
   else if((pid = fork()) == -1)////////////////////////////C
   {
      perror("fork failed");
      exit(1);
   }
   else if(pid == 0)
   {
      printf("Process C : ID - %d\t Parent ID - %d\n",getpid(),getppid());
      if((pid = fork()) == -1)////////////////////////////G
      {
         perror("fork failed");
         exit(1);
      }
      if(pid == 0)
      {
         printf("Process G : ID - %d\t Parent ID - %d\n",getpid(),getppid());
      }
   }
   while(wait(NULL)!=-1);//parent doesn't exit till completion of all child(-1-all child terminated)
   return 0;
}
```

OUTPUT

```
Process A : ID - 24105   Parent ID - 24104
Process C : ID - 24110   Parent ID - 24105
Process B : ID - 24109   Parent ID - 24105
Process G : ID - 24111   Parent ID - 24110
Process E : ID - 24113   Parent ID - 24109
Process D : ID - 24112   Parent ID - 24109
Process H : ID - 24115   Parent ID - 24112
Process F : ID - 24114   Parent ID - 24109
Process I : ID - 24116   Parent ID - 24115
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 4.3*/

#include <stdio.h>
#include <stdlib.h>
int n, m, i, j, k,alloc[10][10], max[10][10], avail[10],inst[10],sum[10],f[10];
int ans[10], ind = 0,need[10][10],req[10],x,work[10];
char ch;
void safeseq()
{
    for (k = 0; k<n; k++)
    {
        f[k] = 0;
    }
    for(i=0;i<n;i++)
    {
        work[i] = avail[i];
    }
    for (k = 0; k<n; k++)
    {
        for (i = 0; i<n; i++)
        {
            if (f[i] == 0)
            {
                int flag = 0;
                for (j = 0; j<m; j++)
                {
                    if (need[i][j] > work[j])
                    {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0)
                {
                    ans[ind++] = i;
                    for (int y = 0; y<m; y++)
                        work[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }
    int flag = 1;
    for (i = 0; i<n; i++)
    {
        if (f[i] == 0)
        {
            flag = 0;
            printf("\nThe system is not safe..\n");
            break;
        }
    }
    if (flag == 1)
    {
        printf("\nThe SAFE Sequence is:\n");
        for (i = 0; i<n - 1; i++)
            printf(" P%d->", ans[i]);
        printf("P%d", ans[n - 1]);
    }
    printf("\n");
}
int main()
{
    printf("\nEnter the number of processes:\n");
    scanf("%d", &n);
    printf("Enter the number of resource types\n");
    scanf("%d",&m);
    for (i = 0; i<m; i++)
    {
        printf("Enter the number of instances of resource %d\n",i+1);
        scanf("%d",&inst[i]);
    }
    printf("\nEnter values for the allocation matrix;\n");
    for (i = 0; i<n; i++)
    {
        for (j = 0; j<m; j++)
        {
            scanf("%d", &alloc[i][j]);
        }
    }
    printf("\nEnter values for the max matrix;\n");
    for (i = 0; i<n; i++)
    {
        for (j = 0; j<m; j++)
        {
            scanf("%d", &max[i][j]);
        }
    }
    printf("\n\nNEED MATRIX:\n");
    for (i = 0; i<n; i++)
    {
        for (j = 0; j<m; j++)
        {
            need[i][j] = max[i][j] - alloc[i][j];
            printf("%d\t", need[i][j]);
        }
        printf("\n");
    }
    sum[0]=0;
    for (i = 0; i<m; i++)
```

```
{
    for (j = 0; j<n; j++)
    {
        sum[i]=sum[i] + alloc[j][i];
    }
}
for (i = 0; i<m; i++)
{
    avail[i] = inst[i] - sum[i];
}
printf("\n\nAVAILABLE VECTOR:\n");
for (i = 0; i<m; i++)
{
    printf("\n%d\t", avail[i]);
}
safeseq();
//Resource request algorithm
printf("\nIs there any request from any process?(y/n)");
scanf("\n%c", &ch);
if (ch == 'n')
exit(0);
else
{
    printf("\nEnter the process ID that needs additional resource:");
    scanf("%d", &x);
    for (i = 0; i<m; i++)
    {
        printf("\nResource request for resource %d :", i + 1);
        scanf("%d", &req[i]);
    }
    printf("\n\nREQUEST VECTOR:\n");
    for (i = 0; i<m; i++)
    {
        printf("%d\t",req[i]);
    }
    int flag1=0,flag2=0;
    for (i = 0; i<m; i++)
    {
        if (req[i] > need[x][i])
            flag1 = 1;
    }
    if(flag1 == 0)
    {
        for (i = 0; i < m; i++)
        {
            if (req[i] > avail[i])
                flag2 = 1;
        }
        if(flag2 == 0)
        {
            for (i = 0; i < m; i++)
            {
                    avail[i] -= req[i];
                    alloc[x][i] += req[i];
                    need[x][i] -= req[i];
                }
            }
            printf("\n\nAVAILABLE VECTOR :\n");
            for (j = 0; j < m; j++)
            printf("%d\t", avail[j]);
            printf("\n\nALLOCATION MATRIX\n");
            for (i = 0; i < n; i++)
            {
                printf("\n");
                for (j = 0; j < m; j++)
                {
                    printf("%d\t", alloc[i][j]);
                }
            }
            printf("\n\nNEED MATRIX:\n");
            for (i = 0; i < n; i++)
            {
                printf("\n");
                for (j = 0; j < m; j++)
                {
                    {
                        need[i][j] = max[i][j] - alloc[i][j];
                        printf("%d\t", need[i][j]);
                    }
                }
            }
            safeseq();
        }
        else
        {
            printf("\nThe request cannot be granted\n");
            return 0;
        }
    }
}
```

OUTPUT

```
Enter the number of processes:
5
Enter the number of resource types
4
Enter the number of instances of resource 1
3
Enter the number of instances of resource 2
17
Enter the number of instances of resource 3
16
Enter the number of instances of resource 4
12
```

```
Enter values for the allocation matrix;
0 1 1 0
1 2 3 1
1 3 6 5
0 6 3 2
0 0 1 4

Enter values for the max matrix;
0 2 1 0
1 6 5 2
2 3 6 6
0 6 5 2
0 6 5 6


NEED MATRIX:
0    1    0    0
0    4    2    1
1    0    0    1
0    0    2    0
0    6    4    2


AVAILABLE VECTOR:

1
5
2
0
The SAFE Sequence is:
 P0-> P3-> P4-> P1->P2

Is there any request from any process?(y/n)y

Enter the process ID that needs additional resource:1

Resource request for resource 1 :0 2 1 0

Resource request for resource 2 :
Resource request for resource 3 :
Resource request for resource 4 :

REQUEST VECTOR:
0    2    1    0

AVAILABLE VECTOR :
1    3    1    0

ALLOCATION MATRIX

0    1    1    0
```

```
1	4	4	1
1	3	6	5
0	6	3	2
0	0	1	4

NEED MATRIX:

0	1	0	0
0	2	1	1
1	0	0	1
0	0	2	0
0	6	4	2
The SAFE Sequence is:
 P0-> P3-> P4-> P1->P2
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 4.1*/

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#define BUFFER_SIZE 5
sem_t empty;
sem_t full;
sem_t mutex;
int buffer[BUFFER_SIZE];
int counter = 0;
void *producer(void *arg)
{
 int item = 0;
 while (1)
 {
 item++;
 sem_wait(&empty);
 sem_wait(&mutex);
 buffer[counter] = item;
 counter++;
 printf("Producer produced item: %d\n", item);
 sem_post(&mutex);
 sem_post(&full);
 }
}
void *consumer(void *arg)
{
 int item;
 while (1)
 {
 sem_wait(&full);
 sem_wait(&mutex);
 counter--;
 item = buffer[counter];
 printf("Consumer consumed item: %d\n", item);
 sem_post(&mutex);
 sem_post(&empty);
 }
}
int main()
{
 pthread_t producer_thread, consumer_thread;
 sem_init(&empty, 0, BUFFER_SIZE);
 sem_init(&full, 0, 0);
 sem_init(&mutex, 0, 1);
 pthread_create(&producer_thread, NULL, producer, NULL);
 pthread_create(&consumer_thread, NULL, consumer, NULL);
 pthread_join(producer_thread, NULL);
 pthread_join(consumer_thread, NULL);
 sem_destroy(&empty);
 sem_destroy(&full);
 sem_destroy(&mutex);
 return 0;
}
```

OUTPUT

```
Producer produced item: 1
Producer produced item: 2
Producer produced item: 3
Consumer consumed item: 3
Producer produced item: 4
Consumer consumed item: 4
Producer produced item: 5
Consumer consumed item: 5
Producer produced item: 6
Consumer consumed item: 6
Producer produced item: 7
Consumer consumed item: 7
Producer produced item: 8
Consumer consumed item: 8
...
...
...
...
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 6.1*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <limits.h>

void fcfs(int head, int req[], int reqno)
{
    printf("\nFCFS Disk Scheduling:\n");
    int tmvt = 0;
    for (int i = 0; i < reqno; i++)
    {
        printf("%d ", req[i]);
        tmvt += abs(head - req[i]);
        head = req[i];
    }
    printf("\nTotal Head Movement: %d\n\n", tmvt);
}
void scan(int head, int req[], int reqno, int dsize)
{
    printf("SCAN Disk Scheduling:\n");
    int tmvt = 0;
    int direction = 1; // 1-movetohigherpositions,-1-lower
    for (int i = 0; i < reqno - 1; i++)
    {
        for (int j = 0; j < reqno - i - 1; j++)
        {
            if (req[j] > req[j + 1])
            {
                int temp = req[j];
                req[j] = req[j + 1];
                req[j + 1] = temp;
            }
        }
    }
    int i;
    for (i = 0; i < reqno; i++)
    {
        if (req[i] >= head)
        {
            break;
        }
    }
    int current = i;
    for (; current < reqno; current++)
    {
        printf("%d ", req[current]);
        tmvt += abs(head - req[current]);
        head = req[current];
    }
    printf("%d ", dsize - 1);
    tmvt += abs(head - (dsize - 1));
    head = dsize - 1;
    for (current = i - 1; current >= 0; current--)
    {
        printf("%d ", req[current]);
        tmvt += abs(head - req[current]);
        head = req[current];
    }
    printf("\nTotal Head Movement: %d\n\n", tmvt);
}
void cscan(int head, int req[], int reqno, int dsize)
{
    printf("C-SCAN Disk Scheduling:\n");
    int tmvt = 0;
    for (int i = 0; i < reqno - 1; i++)
    {
        for (int j = 0; j < reqno - i - 1; j++)
        {
            if (req[j] > req[j + 1])
            {
                int temp = req[j];
                req[j] = req[j + 1];
                req[j + 1] = temp;
            }
        }
    }
    int i;
    for (i = 0; i < reqno; i++)
    {
        if (req[i] >= head)
        {
            break;
        }
    }
    int current = i;
    for (; current < reqno; current++)
    {
        printf("%d ", req[current]);
        tmvt += abs(head - req[current]);
        head = req[current];
    }
    printf("%d ", dsize - 1);
    tmvt += abs(head - (dsize - 1));
    head = dsize - 1;
    printf("0 ");
    tmvt += abs(head - 0);
    head = 0;
    for (current = 0; current < i; current++)
    {
        printf("%d ", req[current]);
        tmvt += abs(head - req[current]);
        head = req[current];
    }
    printf("\nTotal Head Movement: %d\n\n", tmvt);
}
void look(int head, int req[], int reqno)
{
    printf("LOOK Disk Scheduling:\n");
    int tmvt = 0;
    int direction = 1;
    for (int i = 0; i < reqno - 1; i++)
    {
        for (int j = 0; j < reqno - i - 1; j++)
        {
            if (req[j] > req[j + 1])
            {
                int temp = req[j];
                req[j] = req[j + 1];
                req[j + 1] = temp;
            }
        }
    }
    int i;
    for (i = 0; i < reqno; i++)
    {
        if (req[i] >= head)
        {
            break;
        }
    }
    int current = i;
    for (; current < reqno; current++)
    {
        printf("%d ", req[current]);
        tmvt += abs(head - req[current]);
        head = req[current];
    }
    direction = -1;
    for (current = i - 1; current >= 0; current--)
    {
        printf("%d ", req[current]);
        tmvt += abs(head - req[current]);
        head = req[current];
    }
    printf("\nTotal Head Movement: %d\n\n", tmvt);
}
void sstf(int head, int req[], int reqno)
{
    printf("SSTF Disk Scheduling:\n");
    int tmvt = 0;
    bool visited[reqno];
    for (int i = 0; i < reqno; i++)
        visited[i] = false;
    int mindist, index;
    for (int i = 0; i < reqno; i++)
    {
        mindist = INT_MAX;
        for (int j = 0; j < reqno; j++)
        {
            if (!visited[j] && abs(head - req[j]) <= mindist)
            {
                mindist = abs(head - req[j]);
                index = j;
            }
        }
        visited[index] = true;
        printf("%d ", req[index]);
        tmvt += mindist;
        head = req[index];
    }
    printf("\nTotal Head Movement: %d\n\n", tmvt);
}
int main()
{
    int head, dsize, reqno;
    printf("Enter the total number of disk requests: ");
    scanf("%d", &reqno);
    int req[reqno];
    printf("Enter the disk requests: ");
    for (int i = 0; i < reqno; i++)
        scanf("%d", &req[i]);
    printf("Enter the initial head position: ");
    scanf("%d", &head);
    printf("Enter the disk size: ");
    scanf("%d", &dsize);
    fcfs(head, req,reqno);
    scan(head, req,reqno, dsize);
    cscan(head, req,reqno, dsize);
    look(head, req,reqno);
    sstf(head, req,reqno);
    return 0;
}
```

OUTPUT

```
Enter the total number of disk requests: 8
Enter the disk requests: 98 183 41 122 14 124 65 67
Enter the initial head position: 53
Enter the disk size: 200

FCFS Disk Scheduling:
98 183 41 122 14 124 65 67
```

```
Total Head Movement: 632

SCAN Disk Scheduling:
65 67 98 122 124 183 199 41 14
Total Head Movement: 331

C-SCAN Disk Scheduling:
65 67 98 122 124 183 199 0 14 41
Total Head Movement: 386

LOOK Disk Scheduling:
65 67 98 122 124 183 41 14
Total Head Movement: 299

SSTF Disk Scheduling:
65 67 41 14 98 122 124 183
Total Head Movement: 236
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 3.1*/

#include <stdio.h>
#include <stdlib.h>
struct Process {
int id;
int bt;
int wt;
int tat;
};
void fcfs(struct Process p[], int n);
void sjf(struct Process p[], int n);
int main()
{
   int n, i, choice;
   printf("Enter the number of processes involved: ");
   scanf("%d", &n);
   struct Process p[n];
   printf("Enter the burst time of each of the processes:\n");
   for (i = 0; i < n; i++)
   {
      p[i].id = i + 1;
      printf("Process %d: ", p[i].id);
      scanf("%d", &p[i].bt);
      p[i].wt = p[i].tat = 0;
   }
   L:
      printf("\nSelect the CPU Scheduling Algorithm:\n");
      printf("1. FCFS\n2. SJF\n3. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
               fcfs(p, n);
               goto L;
         case 2:
               sjf(p, n);
               goto L;
         case 3:
               exit(0);
      }
      return 0;
}
void fcfs(struct Process p[], int n)
{
   int i, j, current_time = 0;
   float awt = 0, atat = 0, throughput;
```

```
    printf("\nFCFS Scheduling Algorithm:\n\n");
    printf("Gantt Chart:\n");
    printf("-----------\n");
    printf("0");
    for (i = 0; i < n; i++)
    {
        p[i].wt = current_time;
        p[i].tat = p[i].wt + p[i].bt;
        awt += p[i].wt;
        atat += p[i].tat;
        current_time = p[i].tat;
        printf("| P%d | %d ", p[i].id, current_time);
    }
    printf("\n\n");
    awt /= n;
    atat /= n;
    throughput = n / (float) current_time;
    printf("Process\tBurst Time\tWaiting Time\tTurn Around Time\n");
    printf("-------\t----------\t------------\t---------------\n");
    for (i = 0; i < n; i++)
    {
        printf("P%d\t%d\t\t%d\t\t%d\n", p[i].id, p[i].bt, p[i].wt, p[i].tat);
    }
    printf("\nAverage Waiting Time: %.2f", awt);
    printf("\nAverage Turn Around Time: %.2f", atat);
    printf("\nThroughput: %.2f processes per unit time\n", throughput);
}
void sjf(struct Process p[], int n)
{
    int i, j, current_time = 0, min;
    float awt = 0, atat = 0, throughput;
    struct Process temp;
    printf("\nSJF Scheduling Algorithm:\n\n");
    for (i = 0; i < n; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (p[i].bt > p[j].bt)
            {
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
    printf("Gantt Chart:\n");
    printf("-----------\n");
    printf("0");
    for (i = 0; i < n; i++)
    {
        p[i].wt = current_time;
        p[i].tat = p[i].wt + p[i].bt;
        awt += p[i].wt;
        atat += p[i].tat;
        current_time = p[i].tat;
        printf("| P%d | %d ", p[i].id, current_time);
    }
    printf("\n\n");
    awt /= n;
    atat /= n;
    throughput = n / (float) current_time;
    printf("Process\tBurst Time\tWaiting Time\tTurn Around Time\n");
    printf("-------\t----------\t------------\t---------------\n");
    for (i = 0; i < n; i++)
    {
        printf("P%d\t%d\t\t%d\t\t%d\n", p[i].id, p[i].bt, p[i].wt, p[i].tat);
    }
    printf("\nAverage Waiting Time: %.2f", awt);
    printf("\nAverage Turn Around Time: %.2f", atat);
    printf("\nThroughput: %.2f processes per unit time\n", throughput);
}
```

OUTPUT

```
Enter the number of processes involved: 4
Enter the burst time of each of the processes:
Process 1: 6
Process 2: 8
Process 3: 7
Process 4: 3

Select the CPU Scheduling Algorithm:
1. FCFS
2. SJF
3. Exit
Enter your choice: 1

FCFS Scheduling Algorithm:

Gantt Chart:
-----------
0| P1 | 6 | P2 | 14 | P3 | 21 | P4 | 24

Process Burst Time      Waiting Time    Turn Around Time
------- ----------      ------------    ---------------
P1      6               0               6
P2      8               6               14
P3      7               14               21
P4      3               21               24

Average Waiting Time: 10.25
Average Turn Around Time: 16.25
Throughput: 0.17 processes per unit time

Select the CPU Scheduling Algorithm:
```

```
1. FCFS
2. SJF
3. Exit
Enter your choice: 2

SJF Scheduling Algorithm:

Gantt Chart:
-----------
0| P4 | 3 | P1 | 9 | P3 | 16 | P2 | 24

Process Burst Time      Waiting Time    Turn Around Time
------- ----------      ------------    ---------------
P4      3               0               3
P1      6               3               9
P3      7               9               16
P2      8               16               24

Average Waiting Time: 7.00
Average Turn Around Time: 13.00
Throughput: 0.17 processes per unit time

Select the CPU Scheduling Algorithm:
1. FCFS
2. SJF
3. Exit
Enter your choice: 3
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 3.2*/

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

typedef struct process {
    int pid;
    int at;
    int bt;
    int rmt;
    int ct;
    int wt;
    int rst;
    int tat;
} Process;

void sjf_nonpreemptive(Process *p, int n);
void sjf_preemptive(Process *p, int n);
void swap(Process *a, Process *b);
void print_results(Process *p, int n, float awt, float atat, float art, float throughput);

int main()
{
    int n, i, choice;
    float awt = 0, atat = 0, art = 0, throughput;
    Process *p;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    p = (Process*)malloc(n * sizeof(Process));
    printf("\nEnter the arrival time and burst time for each process:\n");
    for (i = 0; i < n; i++)
    {
        printf("\nProcess %d:\n", i + 1);
        p[i].pid = i + 1;
        printf("Arrival time: ");
        scanf("%d", &p[i].at);
        printf("Burst time: ");
        scanf("%d", &p[i].bt);
        p[i].rmt = p[i].bt;
    }
    for (i = 0; i < n; i++)
    {
        p[i].ct = -1;
    }
    while (1)
    {
        printf("\nSelect the SJF algorithm:\n");
        printf("1. Non-preemptive\n2. Preemptive\n3. Exit\n");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                sjf_nonpreemptive(p, n);
                break;
            case 2:
                sjf_preemptive(p, n);
                break;
            case 3:
                free(p);
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
        awt = 0, atat = 0, art = 0;
        for (i = 0; i < n; i++)
        {
            p[i].ct = p[i].at + p[i].wt + p[i].bt;
            p[i].tat = p[i].ct - p[i].at;
            p[i].rst = p[i].wt;
            awt += p[i].wt;
            atat += p[i].tat;
            art += p[i].rst;
        }
        awt /= n;
        atat /= n;
        art /= n;
        throughput = 0;
        for (i = 0; i < n; i++)
        {
            throughput += p[i].bt;
        }
        float throughputt = n / throughput;
        print_results(p, n, awt, atat, art, throughputt);
    }
    return 0;
}
void sjf_nonpreemptive(Process *p, int n)
{
    int i, j;
    int current_time = 0;
    int completed = 0;
    while (completed < n)
    {
        int min_bt = INT_MAX;
        int min_idx = -1;
        for (i = 0; i < n; i++)
        {
            if (p[i].at <= current_time && p[i].ct == -1 && p[i].bt < min_bt)
            {
                min_bt = p[i].bt;
                min_idx = i;
            }
            else if (p[i].at <= current_time && p[i].ct == -1 && p[i].bt == min_bt)
            {
                if (p[i].at < p[min_idx].at) //tie-select early arrived
                {
                    min_idx = i;
                }
            }
        }
        if (min_idx == -1)
        {
            current_time = p[completed].at;//none arrived
        }
        else
        {
            p[min_idx].ct = current_time + p[min_idx].bt;
            p[min_idx].wt = p[min_idx].ct - p[min_idx].at - p[min_idx].bt;
            current_time = p[min_idx].ct;
            completed = completed + 1;
        }
    }
}
void sjf_preemptive(Process *p, int n)
{
    int i, j, t, min_idx;
    int *rmt = malloc(n * sizeof(int));
    for (i = 0; i < n; i++)
    {
        rmt[i] = p[i].bt;
    }
    for (t = 0;; t++)
    {
        min_idx = -1;
        for (i = 0; i < n; i++)
        {
            if (p[i].at <= t && rmt[i] > 0)
            {
                if (min_idx == -1 || rmt[i] < rmt[min_idx])
                {
                    min_idx = i;
                }
            }
        }
        if (min_idx == -1)
        {
            break;
        }
        rmt[min_idx]--;
        if (rmt[min_idx] == 0)
        {
            p[min_idx].wt = t - p[min_idx].bt - p[min_idx].at + 1;
            if (p[min_idx].wt < 0)
            {
                p[min_idx].wt = 0;
            }
        }
    }
    free(rmt);
}
void swap(Process *a, Process *b)
{
    Process temp = *a;
    *a = *b;
    *b = temp;
}
void print_results(Process *p, int n, float awt, float atat, float art, float throughput)
{
    int i;
    printf("\nPID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\tResponse Time\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n", p[i].pid, p[i].at, p[i].bt, p[i].wt, p[i].tat, p[i].rst);
    }
    printf("\nAverage waiting time = %f\n", awt);
    printf("Average turnaround time = %f\n", atat);
    printf("Average response time = %f\n", art);
    printf("Throughput = %f processes per unit time\n", throughput);
}
```

OUTPUT

```
Enter the number of processes: 6

Enter the arrival time and burst time for each process:

Process 1:
Arrival time: 0
Burst time: 8

Process 2:
Arrival time: 1
Burst time: 4

Process 3:
Arrival time: 2
Burst time: 2

Process 4:
Arrival time: 3
Burst time: 1

Process 5:
Arrival time: 4
```

```
Burst time: 3

Process 6:
Arrival time: 5
Burst time: 2

Select the SJF algorithm:
1. Non-preemptive
2. Preemptive
3. Exit
1

PID     Arrival Time    Burst Time      Waiting Time    Turnaround Time Response Time
1        0              8               0                8              0
2        1              4               15              19              15
3        2              2               7                9               7
4        3              1               5                6               5
5        4               3               9              12               9
6        5               2               6               8               6

Average waiting time = 7.000000
Average turnaround time = 10.333333
Average response time = 7.000000
Throughput = 0.300000 processes per unit time

Select the SJF algorithm:
1. Non-preemptive
2. Preemptive
3. Exit
2

PID     Arrival Time    Burst Time      Waiting Time    Turnaround Time Response Time
1       0               8               12               20             12
2       1               4                5              9               5
3       2               2                0              2               0
4       3               1                1              2               1
5       4               3                6              9               6
6       5               2                0              2               0

Average waiting time = 4.000000
Average turnaround time = 7.333333
Average response time = 4.000000
Throughput = 0.300000 processes per unit time

Select the SJF algorithm:
1. Non-preemptive
2. Preemptive
3. Exit
3
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 3.3*/

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
typedef struct process {
    int pid;
    int at;
    int bt;
    int rmt;
    int ct;
    int wt;
    int rst;
    int tat;
    int pt;
} Process;
void prio_nonpreemptive(Process *p, int n);
void prio_preemptive(Process *p, int n);
void swap(Process *a, Process *b);
void print_results(Process *p, int n, float awt, float atat, float art, float throughput);

int main()
{
    int n, i, choice;
    float awt = 0, atat = 0, art = 0, throughput;
    Process *p;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    p = (Process *)malloc(n * sizeof(Process));
    printf("\nEnter the arrival time, burst time, and priority for each process:\n");
    for (i = 0; i < n; i++)
    {
        printf("\nProcess %d:\n", i + 1);
        p[i].pid = i + 1;
        printf("Arrival time: ");
        scanf("%d", &p[i].at);
        printf("Burst time: ");
        scanf("%d", &p[i].bt);
        p[i].rmt = p[i].bt;
        printf("Priority: ");
        scanf("%d", &p[i].pt);
    }
    for (i = 0; i < n; i++)
    {
        p[i].ct = -1;
    }
    while (1)
    {
        printf("\nSelect the SJF algorithm:\n");
        printf("1. Non-preemptive\n2. Preemptive\n3. Exit\n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            prio_nonpreemptive(p, n);
            break;
        case 2:
            prio_preemptive(p, n);
            break;
        case 3:
            free(p);
            return 0;
        default:
            printf("Invalid choice. Please try again.\n");
        }
        awt = 0, atat = 0, art = 0;
        for (i = 0; i < n; i++)
        {
            p[i].ct = p[i].at + p[i].wt + p[i].bt;
            p[i].tat = p[i].ct - p[i].at;
            p[i].rst = p[i].wt;
            awt += p[i].wt;
            atat += p[i].tat;
            art += p[i].rst;
        }
        awt /= n;
        atat /= n;
        art /= n;
        throughput = 0;
        for (i = 0; i < n; i++)
        {
            throughput += p[i].bt;
        }
        float throughputt = n / throughput;
        print_results(p, n, awt, atat, art, throughputt);
    }
    return 0;
}
void prio_nonpreemptive(Process *p, int n)
{
    int i, j;
    int current_time = 0;
    int completed = 0;
    printf("Gantt chart");
    printf("\n0 - ");
    while (completed < n)
    {
        int min_pt = INT_MAX;
        int min_idx = -1;
        for (i = 0; i < n; i++)
        {
            if (p[i].at <= current_time && p[i].ct == -1 && p[i].pt < min_pt)
            {
                min_pt = p[i].pt;
                min_idx = i;
            } else if (p[i].at <= current_time && p[i].ct == -1 && p[i].pt == min_pt)
            {
                if (p[i].at < p[min_idx].at) // tie-select early arrived
                {
                    min_idx = i;
                }
            }
        }
        if (min_idx == -1)
        {
            current_time = p[completed].at; // none arrived
        }
        else
        {
            p[min_idx].ct = current_time + p[min_idx].bt;
            printf("P%d - %d - ", p[min_idx].pid, p[min_idx].ct);
            p[min_idx].wt = p[min_idx].ct - p[min_idx].at - p[min_idx].bt;
            current_time = p[min_idx].ct;
            completed = completed + 1;
        }
    }
}
void prio_preemptive(Process *p, int n)
{
    int i, j, t, min_idx;
    int *rmt = malloc(n * sizeof(int));
    printf("Gantt chart");
    printf("\n0 - ");
    for (i = 0; i < n; i++)
    {
        rmt[i] = p[i].bt;
    }
    for (t = 0;; t++)
    {
        min_idx = -1;
        for (i = 0; i < n; i++)
        {
            if (p[i].at <= t && rmt[i] > 0)
            {
                if (min_idx == -1 || p[i].pt < p[min_idx].pt)
                {
                    min_idx = i;
                }
            }
        }
        if (min_idx == -1)
        {
            break;
        }
        rmt[min_idx]--;
        printf("P%d - %d - ", p[min_idx].pid,t+1);
        if (rmt[min_idx] == 0)
        {
            p[min_idx].wt = t - p[min_idx].bt - p[min_idx].at + 1;
            if (p[min_idx].wt < 0)
            {
                p[min_idx].wt = 0;
            }
        }
    }
    free(rmt);
}
void swap(Process *a, Process *b)
{
    Process temp = *a;
    *a = *b;
    *b = temp;
}
void print_results(Process *p, int n, float awt, float atat, float art, float throughput)
{
    int i;
    printf("\n");
    printf("\nPID\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n", p[i].pid, p[i].at, p[i].bt, p[i].pt, p[i].wt, p[i].tat);
    }
    printf("\nAverage waiting time = %f\n", awt);
    printf("Average turnaround time = %f\n", atat);
    printf("Throughput = %f processes per unit time\n", throughput);
}
```

OUTPUT

```
Enter the number of processes: 5

Enter the arrival time, burst time, and priority for each process:

Process 1:
Arrival time: 0
Burst time: 4
Priority: 4

Process 2:
Arrival time: 1
Burst time: 3
Priority: 3

Process 3:
Arrival time: 2
```

```
Burst time: 1
Priority: 2

Process 4:
Arrival time: 3
Burst time: 5
Priority: 1

Process 5:
Arrival time: 4
Burst time: 2
Priority: 1

Select the SJF algorithm:
1. Non-preemptive
2. Preemptive
3. Exit
1
Gantt chart
0 - P1 - 4 - P4 - 9 - P5 - 11 - P3 - 12 - P2 - 15 -

PID    Arrival Time    Burst Time      Priority        Waiting Time    Turnaround Time
1      0         4         4         0            4
2      1         3         3         11            14
3      2         1         2         9            10
4      3         5         1         1            6
5      4         2         1         5            7

Average waiting time = 5.200000
Average turnaround time = 8.200000
Throughput = 0.333333 processes per unit time

Select the SJF algorithm:
1. Non-preemptive
2. Preemptive
3. Exit
2
Gantt chart
0 - P1 - 1 - P2 - 2 - P3 - 3 - P4 - 4 - P4 - 5 - P4 - 6 - P4 - 7 - P4 - 8 - P5 - 9 - P5 - 10 - P2 - 11 - P2 -
12 - P1 - 13 - P1 - 14 - P1 - 15 -

PID    Arrival Time    Burst Time      Priority        Waiting Time    Turnaround Time
1      0         4         4         11            15
2      1         3         3         8            11
3      2         1         2         0            1
4      3         5         1         0            5
5      4         2         1         4            6

Average waiting time = 4.600000
Average turnaround time = 7.600000
Throughput = 0.333333 processes per unit time
```

```
Select the SJF algorithm:
1. Non-preemptive
2. Preemptive
3. Exit
3
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 3.4*/

#include <stdio.h>
#include <stdbool.h>
#define MAX_PROCESSES 10
typedef struct {
    int pid;
    int bt;
    int rt;
    int at;
    int wt;
    int tat;
} Process;

int main()
{
    int n;
    Process pro[MAX_PROCESSES];
    printf("Enter the number of processes (up to %d): ", MAX_PROCESSES);
    scanf("%d", &n);
    printf("Enter the arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++)
    {
        printf("Process %d: ", i + 1);
        scanf("%d %d", &pro[i].at, &pro[i].bt);
        pro[i].pid = i + 1;
        pro[i].rt = pro[i].bt;
    }
    int time_quantum;
    printf("\nSelect the time quantum (in ms) from the following:\n");
    printf("1. 2 ms\n2. 4 ms\n3. 5 ms\n4. 8 ms\n5. 10 ms\n");
    printf("Enter your choice (1-5): ");
    int choice;
    scanf("%d", &choice);

    switch (choice)
    {
        case 1:
            time_quantum = 2;
            break;
        case 2:
            time_quantum = 4;
            break;
        case 3:
            time_quantum = 5;
            break;
        case 4:
            time_quantum = 8;
            break;
        case 5:
            time_quantum = 10;
            break;
        default:
            printf("Invalid choice. Exiting.\n");
            return 1;
    }
    int current_time = 0;
    int completed_processes = 0;
    bool is_completed[MAX_PROCESSES] = {false};
    int twt = 0;
    int ttat = 0;
    printf("\nGantt Chart:\n");
    while (completed_processes < n)
    {
        for (int i = 0; i < n; i++)
        {
            if (!is_completed[i])
            {
                //printf("| P%d ", processes[i].process_id);
                int execution_time = (pro[i].rt <= time_quantum) ? pro[i].rt : time_quantum;
                current_time += execution_time;
                pro[i].rt -= execution_time;
                printf("(%d-P%d-%d)|", current_time - execution_time,pro[i].pid, current_time);
                if (pro[i].rt <= 0)
                {
                    pro[i].tat = current_time - pro[i].at;
                    pro[i].wt = pro[i].tat - pro[i].bt;
                    twt += pro[i].wt;
                    ttat += pro[i].tat;
                    is_completed[i] = true;
                    completed_processes++;
                }
            }
        }
    }
    printf("\nProcess\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++)
    {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", pro[i].pid, pro[i].bt,
                 pro[i].at, pro[i].wt, pro[i].tat);
    }
    double awt = (double)twt / n;
    double atat = (double)ttat / n;
    double throughput = (double)n / current_time;
    printf("\nAverage Waiting Time: %.2lf\n", awt);
    printf("Average Turnaround Time: %.2lf\n", atat);
    printf("Throughput: %.2lf processes/ms\n", throughput);
    return 0;
}
```

OUTPUT

```
Enter the number of processes (up to 10): 5
Enter the arrival time and burst time for each process:
Process 1: 0 5
Process 2: 1 3
Process 3: 2 1
Process 4: 3 2
Process 5: 4 3

Select the time quantum (in ms) from the following:
1. 2 ms
2. 4 ms
3. 5 ms
4. 8 ms
5. 10 ms
Enter your choice (1-5): 1

Gantt Chart:
(0-P1-2)|(2-P2-4)|(4-P3-5)|(5-P4-7)|(7-P5-9)|(9-P1-11)|(11-P2-12)|(12-P5-13)|(13-P1-14)|
Process Burst Time      Arrival Time    Waiting Time    Turnaround Time
1       5               0               9               14
2       3               1               8               11
3       1               2               2               3
4       2               3               2               4
5       3               4               6               9

Average Waiting Time: 5.40
Average Turnaround Time: 8.20
Throughput: 0.36 processes/ms
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 5.2*/

#include <stdio.h>
#include <stdlib.h>
#include<stdbool.h>
struct pagetable {
	int num;
	int index;
};
struct pages {
	int num;
	int count;
};
void fifo(int m, int n, int pages[])
{
	printf("\nFIFO\n");
	struct pagetable table[m];
	int index = 0, free = 1, faults = 0;
	for(int i=0; i<m; i++)
		table[i].num = -1;
	for (int i=0; i<n; i++)
	{
		printf("%d: ", pages[i]);
		int contains = 0;
		for (int j=0; j<m; j++)
			if (table[j].num == pages[i])
			{
				contains = 1;
				break;
			}
		if (contains)
		{
			for (int j=0; j<m; j++)
			{
				if (free)
				{
					if (j < index)
						printf("%d ", table[j].num);
					else
						printf(" ");
				} else
					printf("%d ", table[j].num);
			}
			printf("\n");
		}
		else
		{
			table[index].num = pages[i];
```

```
            index = (index + 1) % m;
            faults++;
            if (index == 0)
                free = 0;
            for (int j=0; j<m; j++)
            {
                if (free)
                {
                    if (j < index)
                        printf("%d ", table[j].num);
                    else
                        printf(" ");
                } else
                    printf("%d ", table[j].num);
            }
            printf("\n");
        }
    }
    printf("\nNo of page faults = %d\n", faults);
    printf("Miss ratio = %.2f%%\n",(float)faults/n *100);
    printf("Hit ratio = %.2f%%\n",(float)(n-faults)/n *100);
}
void lru(int m, int n, int pages[])
{
    printf("\nLRU\n");
    struct pagetable table[m];
    int index = -1, free = 1, faults = 0, count = 0;
    for(int i=0; i<m; i++)
        table[i].num = -1;
    for (int i=0; i<n; i++)
    {
        printf("%d: ", pages[i]);
        int contains = 0;
        for (int j=0; j<m; j++)
            if (table[j].num == pages[i])
            {
                table[j].index = count;
                count++;
                for (int j=0; j<m; j++)
                {
                    if (free)
                    {
                        if (j <= index)
                            printf("%d ", table[j].num);
                        else
                            printf(" ");
                    } else
                        printf("%d ", table[j].num);
                }
                printf("\n");
                contains = 1;
            }
        if (contains == 0)
        {
            if (free)
            {
                index = (index + 1) % m;
                if (index == (m-1))
                    free = 0;
            }
            else
            {
                index = 0;

                for (int j=1; j<m; j++)
                    if (table[j].index < table[index].index)
                        index = j;
            }
            table[index].num = pages[i];
            table[index].index = count;
            count++;
            faults++;
            for (int j=0; j<m; j++)
            {
                if (free)
                {
                    if (j <= index)
                        printf("%d ", table[j].num);
                    else
                        printf(" ");
                } else
                    printf("%d ", table[j].num);
            }
            printf("\n");
        }
    }
    printf("\nNo of page faults = %d\n", faults);
    printf("Miss ratio = %.2f%%\n",(float)faults/n *100);
    printf("Hit ratio = %.2f%%\n",(float)(n-faults)/n *100);
}
void lfu(int m, int n, int pages[])
{
    printf("\nLFU\n");
    struct pagetable table[m];
    struct pages map[n];
    int index = -1, free = 1, faults = 0, count = 0, maplen = 0;
    for(int i=0; i<m; i++)
        table[i].num = -1;
    for (int i=0; i<n; i++)
    {
        printf("%d: ", pages[i]);
        int contains = 0;
        for (int j=0; j<m; j++)
            if (table[j].num == pages[i])
            {
                for (int k=0; k<maplen; k++)
                    if (map[k].num == table[j].num)
                    {
                        map[k].count++;
                        break;
                    }
                table[j].index = count;
                count++;
                for (int j=0; j<m; j++)
                {
                    if (free)
                    {
                        if (j <= index)
                            printf("%d ", table[j].num);
                        else
                            printf(" ");
                    } else
                        printf("%d ", table[j].num);
                }
                printf("\n");
                contains = 1;
            }
        if (contains == 0)
        {
            if (free)
            {
                index = (index + 1) % m;
                if (index == (m-1))
                    free = 0;
            }
            else
            {
                index = 0;
                int index1 = 0, index2 = 0;
                for (int j=1; j<m; j++)
                {
                    for (int k=0; k<maplen; k++)
                        if (map[k].num == table[index].num)
                        {
                            index1 = k;
                            continue;
                        }
                        else if (map[k].num == table[j].num)
                        {
                            index2 = k;
                            continue;
                        }

                    if (map[index2].count < map[index1].count)
                    {
                        index = j;
```

```
                                        }
                                        else if (map[index2].count == map[index1].count)
                                        {
                                                if (table[j].index < table[index].index)
                                                        index = j;
                                        }
                                }
                        }
                        table[index].num = pages[i];
                        int exists = 0;
                        for (int k=0; k<maplen; k++)
                                if (map[k].num == table[index].num)
                                {
                                        map[k].count++;
                                        exists = 1;
                                        break;
                                }
                        if (exists == 0)
                        {
                                map[maplen].num = pages[i];
                                map[maplen].count = 1;
                                maplen++;
                        }
                        table[index].index = count;
                        count++;
                        faults++;
                        for (int j=0; j<m; j++)
                        {
                                if (free)
                                {
                                        if (j <= index)
                                                printf("%d ", table[j].num);
                                        else
                                                printf(" ");
                                }
                                else
                                        printf("%d ", table[j].num);
                        }
                        printf("\n");
                }
        }
        printf("\nNo of page faults = %d\n", faults);
        printf("Miss ratio = %.2f%%\n",(float)faults/n *100);
        printf("Hit ratio = %.2f%%\n",(float)(n-faults)/n *100);
}
void optimal(int frames,int n,int pages[])
{
    int frame[10];
    bool pageFault = false;
    int pageFaultCount = 0;
    int pageHits = 0;
    for (int i = 0; i < frames; i++)
```

```
{
    frame[i] = -1;
}
for (int i = 0; i < n; i++)
{
    int currentPage = pages[i];
    bool pageFound = false;
    for (int j = 0; j < frames; j++)
    {
        if (frame[j] == currentPage)
        {
            pageFound = true;
            pageHits++;
            break;
        }
    }
    if (!pageFound)
    {
        int pageToReplaceIndex = 0;
        int pageToReplaceFarthest = i + 1;
        for (int j = 0; j < frames; j++)
        {
            int nextPageIndex = i + 1;
            while (nextPageIndex < n)
            {
                if (frame[j] == pages[nextPageIndex])
                {
                    break;
                }
                nextPageIndex++;
            }
            if (nextPageIndex == n)
            {
                pageToReplaceIndex = j;
                break;
            }
            else if (nextPageIndex > pageToReplaceFarthest)
            {
                pageToReplaceIndex = j;
                pageToReplaceFarthest = nextPageIndex;
            }
        }
        frame[pageToReplaceIndex] = currentPage;
        pageFaultCount++;
        pageFault = true;
    }
    printf("%d: ", currentPage);
    for (int j = 0; j < frames; j++)
    {
        printf("%d ", frame[j]);
    }
    printf("\n");
    }
    printf("Page Faults: %d\n", pageFaultCount);
    printf("Hit Ratio: %.2f%%\n", (float)pageHits / n * 100);
    printf("Miss Ratio: %.2f%%\n", (float)(n - pageHits) / n * 100);
}
int main()
{
        int m, n, opt;
        printf("Enter the frame capacity: ");
        scanf("%d", &m);
        printf("Enter the no of page requests: ");
        scanf("%d", &n);
        int pages[n];
        printf("Enter the page requests:\n");
        for(int i=0; i<n; i++)
                scanf("%d", &pages[i]);
        while(1)
        {
                printf("\n1. FIFO\n2. LRU\n3. LFU\n4. Optimal\n5. Exit\n");
                printf("Choose option: ");
                scanf("%d", &opt);
                switch (opt)
                {
                        case 1:
                                fifo(m, n, pages);
                                break;
                        case 2:
                                lru(m, n, pages);
                                break;
                        case 3:
                                lfu(m, n, pages);
                                break;
                        case 4:
                                optimal(m, n, pages);
                                break;
                        case 5:
                                printf("\nExit.\n");
                                exit(0);
                        default:
                                printf("\nInvalid option!\n");
                }
        }
        return 0;
}
```

OUTPUT

```
Enter the frame capacity: 4
Enter the no of page requests: 20
Enter the page requests:
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

1. FIFO
```

```
2. LRU
3. LFU
4. Optimal
5. Exit
Choose option: 1

FIFO
1: 1
2: 1 2
3: 1 2 3
4: 1 2 3 4
2: 1 2 3 4
1: 1 2 3 4
5: 5 2 3 4
6: 5 6 3 4
2: 5 6 2 4
1: 5 6 2 1
2: 5 6 2 1
3: 3 6 2 1
7: 3 7 2 1
6: 3 7 6 1
3: 3 7 6 1
2: 3 7 6 2
1: 1 7 6 2
2: 1 7 6 2
3: 1 3 6 2
6: 1 3 6 2

No of page faults = 14
Miss ratio = 70.00%
Hit ratio = 30.00%

1. FIFO
2. LRU
3. LFU
4. Optimal
5. Exit
Choose option: 2

LRU
1: 1
2: 1 2
3: 1 2 3
4: 1 2 3 4
2: 1 2 3 4
1: 1 2 3 4
5: 1 2 5 4
6: 1 2 5 6
2: 1 2 5 6
1: 1 2 5 6
2: 1 2 5 6
3: 1 2 3 6
```

```
7: 1 2 3 7
6: 6 2 3 7
3: 6 2 3 7
2: 6 2 3 7
1: 6 2 3 1
2: 6 2 3 1
3: 6 2 3 1
6: 6 2 3 1

No of page faults = 10
Miss ratio = 50.00%
Hit ratio = 50.00%

1. FIFO
2. LRU
3. LFU
4. Optimal
5. Exit
Choose option: 3

LFU
1: 1
2: 1 2
3: 1 2 3
4: 1 2 3 4
2: 1 2 3 4
1: 1 2 3 4
5: 1 2 5 4
6: 1 2 5 6
2: 1 2 5 6
1: 1 2 5 6
2: 1 2 5 6
3: 1 2 3 6
7: 1 2 3 7
6: 1 2 3 6
3: 1 2 3 6
2: 1 2 3 6
1: 1 2 3 6
2: 1 2 3 6
3: 1 2 3 6
6: 1 2 3 6

No of page faults = 9
Miss ratio = 45.00%
Hit ratio = 55.00%

1. FIFO
2. LRU
3. LFU
4. Optimal
5. Exit
Choose option: 4
```

```
1: 1 -1 -1 -1
2: 1 2 -1 -1
3: 1 2 3 -1
4: 1 2 3 4
2: 1 2 3 4
1: 1 2 3 4
5: 1 2 3 5
6: 1 2 3 6
2: 1 2 3 6
1: 1 2 3 6
2: 1 2 3 6
3: 1 2 3 6
7: 7 2 3 6
6: 7 2 3 6
3: 7 2 3 6
2: 7 2 3 6
1: 1 2 3 6
2: 1 2 3 6
3: 1 2 3 6
6: 1 2 3 6
Page Faults: 8
Hit Ratio: 60.00%
Miss Ratio: 40.00%

1. FIFO
2. LRU
3. LFU
4. Optimal
5. Exit
Choose option: 5

Exit.
```

PROGRAM

```
/*Name : BHAGYA A JAI
Roll number : B21CSB18
Experiment No : 6.2*/

//Program to enter details of students
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#define MAX_STUDENTS 50
typedef struct student {
    char name[50];
    float marks;
} student;

int main() {
    int n;
    printf("Enter the number of students: ");
    scanf("%d", &n);

    if (n <= 0 || n > MAX_STUDENTS) {
        printf("Invalid number of students. Please enter a value between 1 and %d.\n",
MAX_STUDENTS);
        return 1;
    }
    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, sizeof(student) * n, IPC_CREAT | 0666);

    if (shmid == -1) {
        perror("shmget");
        return 1;
    }
    student *students = (student *)shmat(shmid, NULL, 0);
    if (students == (student *)(-1)) {
        perror("shmat");
        return 1;
    }
    for (int i = 0; i < n; i++) {
        printf("Enter details for student %d:\n", i + 1);
        printf("Name: ");
        scanf("%s", students[i].name);
        printf("Marks: ");
        scanf("%f", &students[i].marks);
    }
    shmdt(students);
    return 0;
}
```

```
//Program to find rank
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define MAX_STUDENTS 50
typedef struct student {
    char name[50];
    float marks;
} student;
void calculateRanks(student *students, int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (students[j].marks < students[j + 1].marks) {
                student temp = students[j];
                students[j] = students[j + 1];
                students[j + 1] = temp;
            }
        }
    }
}
void displayRankDetails(student *students, int n) {
    printf("\nRank Details:\n");
    printf("Rank\tName\t\tMarks\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%s\t\t%.2f\n", i + 1, students[i].name, students[i].marks);
    }
}
int main()
{
    int n;
    printf("Enter the number of students: ");
    scanf("%d", &n);
    if (n <= 0 || n > MAX_STUDENTS)
  {
        printf("Invalid number of students. Please enter a value between 1 and %d.\n",
MAX_STUDENTS);
        return 1;
    }
    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, sizeof(student) * n, 0666);

    if (shmid == -1) {
        perror("shmget");
        return 1;
    }
    student *students = (student *)shmat(shmid, NULL, 0);
    if (students == (student *)(-1))
   {
        perror("shmat");
        return 1;
    }
    calculateRanks(students, n);
    displayRankDetails(students, n);
    shmdt(students);
    return 0;
}
```

OUTPUT

```
//First program output
Enter the number of students: 4
Key of shared memory is 0
Enter student details:
Student 1 name: John
Student 1 marks: 87
Student 2 name: Ben
Student 2 marks: 91
Student 3 name: Diya
Student 3 marks: 79
Student 4 name: Isha
Student 4 marks: 93
//Second program output
Rank details of students:
Rank 1: Isha Marks: 93
Rank 2: Ben Marks: 91
Rank 3: John Marks: 87
Rank 4: Diya Marks: 79
```