1. Simulate the following CPU scheduling algorithms

a)FCFS b) SJF c) Priority d) Round Robin

```
#include<stdio.h>
// Remove comments if you execute in Turbo C compiler
//#include<conio.h>

int p[100], pid[100], at[100], bt[100], ct[100], tat[100], wt[100], rt[100];
int n, ch, tq, cpu_idle, Temp_ct;
float avg_tat, avg_wt,t_put, avg_rt;


void P_display()
{
   int i;
   printf("\nprty\tPid\tAt\tBt\tCt\tTat\tWt\trt\n");
   for ( i = 1; i <= n; i++ )
   {

printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n",p[i],pid[i],at[i],bt[i],ct[i],tat[i
],wt[i],rt[i]);
   }
}



void fcfs_data()
{
   int i;
   printf("Enter the no. of processes : ");
   scanf("%d",&n);
   printf("\nEnter the process id's, arrival time and burst times of each
process\n");
   for ( i = 1; i <= n; i++ )
      scanf("%d%d%d",&pid[i], &at[i], &bt[i]);
}
```

```
void fcfs_Sort()
{
    int i,j,temp;
    for ( i = 0; i <= n-1; i++ )
    {
        int flag = 0;
        for ( j = 0; j <= n-1-i; j++ )
        {
            if ( at[j] > at[j+1] )
            {
                flag = 1;
                temp = pid[j];
                pid[j] = pid[j+1];
                pid[j+1] = temp;

                temp = at[j+1];
                at[j+1] = at[j];
                at[j] = temp;

                temp = bt[j+1];
                bt[j+1] = bt[j];
                bt[j] = temp;
            }
        }
        if ( flag == 0 )
            break;
    }
}




void fcfs_computations()
{
    int i;
    for ( i = 1; i <= n; i++)
    {
        if( at[i] > ct[i-1] )
        {
            cpu_idle = at[i] - ct[i-1];
            ct[i] = Temp_ct + bt[i] + cpu_idle;
        }
```

```
        else
        {
            ct[i] = Temp_ct + bt[i];
        }
        Temp_ct = ct[i];
        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt[i];
        rt[i] = wt[i];
    }

    for ( i = 1; i <= n; i++ )
    {
            avg_tat += tat[i];
            avg_wt += wt[i];
        avg_rt += rt[i];
    }

    avg_tat /= n;
    avg_wt /= n;
    avg_rt /= n;
    t_put = ct[n];
    t_put = n/t_put;
}



void sjf_data()
{
    int i;
    printf("Enter the no. of processes : ");
    scanf("%d",&n);
    printf("Here we assume that all processes have arrived at 0\n");
    printf("Enter the process id's and burst times of each process\n");
    for ( i = 1; i <= n; i++ )
    {
         scanf("%d",&pid[i]);
         scanf("%d",&bt[i]);
    }
}
```

```
void sjf_sort()
{
    int i,j,temp;
    for ( i = 0; i <= n-1; i++ )
    {
        int flag = 0;
        for ( j = 0; j <= n-1-i; j++ )
        {
            if ( bt[j] > bt[j+1] )
            {
                flag = 1;
                temp = pid[j];
                pid[j] = pid[j+1];
                pid[j+1] = temp;

                temp = bt[j+1];
                bt[j+1] = bt[j];
                bt[j] = temp;
            }
        }
        if ( flag == 0 )
            break;
    }
}



void sjf_computations()
{
    int i;
    for ( i = 1; i <= n; i++)
    {
        ct[i] = bt[i] + tat[i-1];
        tat[i] = ct[i] ; // + at[i] ==> all processes arrived at 0
        wt[i] = tat[i] - bt[i];
        rt[i] = wt[i];
    }

    for ( i = 1; i <= n; i++ )
    {
            avg_tat += tat[i];
            avg_wt += wt[i];
        avg_rt += rt[i];
    }
```

```
    avg_tat /= n;
    avg_wt /= n;
    avg_rt /= n;
    t_put = ct[n];
    t_put = n/t_put;

}



void priority_data()
{
    int i;
    printf("Enter the no. of processes : ");
    scanf("%d",&n);
    printf("Here we assume that all processes have arrived at 0\n");
    printf("Lower the number higher the priority\n");
    printf("Enter the priority, process id's and burst times of each
process\n");
    for ( i = 1; i <= n; i++ )
    {
        scanf("%d",&p[i]);
        scanf("%d",&pid[i]);
        scanf("%d",&bt[i]);
    }
}

void priority_sort()
{
    int i,j,temp;
    for ( i = 0; i <= n-1; i++ )
    {
        int flag = 0;
        for ( j = 0; j <= n-1-i; j++ )
        {
            if ( p[j] > p[j+1] )
            {
                flag = 1;
                temp = p[j];
                p[j] = p[j+1];
                p[j+1] = temp;

                temp = pid[j+1];
                pid[j+1] = pid[j];
                pid[j] = temp;
```

```
                temp = bt[j+1];
                bt[j+1] = bt[j];
                bt[j] = temp;
            }
        }
        if ( flag == 0 )
            break;
    }
}


void priority_computations()
{
    int i;
    for ( i = 1; i <= n; i++)
    {
        ct[i] = bt[i] + tat[i-1];
        tat[i] = ct[i] ; // + at[i] ==> all processes arrived at 0
        wt[i] = tat[i] - bt[i];
        if ( ch == 3 )
            rt[i] = wt[i];
        else
        {
            if ( i == 1 )
                rt[i] = 0;
            else
                rt[i] = ct[i-1] + 1;
        }
    }

    for ( i = 1; i <= n; i++ )
    {
             avg_tat += tat[i];
             avg_wt += wt[i];
        avg_rt += rt[i];
    }

    avg_tat /= n;
    avg_wt /= n;
    avg_rt /= n;
    t_put = ct[n];
    t_put = n/t_put;
}
```

```
void RR_data()
{
    int i;
    printf("Enter the No. of processes : ");
    scanf("%d", &n);

    printf(" -------- \n");
    printf("| NOTE : | Here we are assuming all the processes have arrived at
the Time stamp 0\n");
    printf(" -------- \n");
    printf("Enter the burst times of the process\n");
    for ( i = 0; i < n; i++ )
        scanf("%d", &bt[i]);
    printf("Enter Time Quantum : ");
    scanf("%d", &tq);
}


void RR_computations()
{
    int i, currentTime = 0, completed = 0;
    int remainingTime[100];
    for ( i = 0; i < n; i++ )
        remainingTime[i] = bt[i];

    while( completed < n )
    {
        for( i = 0; i < n; i++ )
        {
            if( remainingTime[i] > 0 )
            {
                int exeTime = (remainingTime[i] < tq) ? remainingTime[i] : tq;
                currentTime += exeTime;
                remainingTime[i] -= exeTime;

                if ( remainingTime[i] == 0 )
                {
                    completed++;
                    wt[i] = currentTime - bt[i];
                    tat[i] = currentTime;
                }
            }
        }
    }
```

```
    for ( i = 0; i < n; i++ )
    {
        avg_wt += wt[i];
        avg_tat += tat[i];
    }

    avg_wt /= n;
    avg_tat /= n;

    // Displaying the Computed data
    printf("Bt\t tat\t wt\n");
    for ( i = 0; i < n; i++ )
        printf("%d\t %d\t %d\n",bt[i], tat[i], wt[i]);
    printf("Average Waiting Time: %f\n", avg_tat);
    printf("Average Turnaround Time: %f\n", avg_wt);
}


void final_res()
{
    printf("\n\nThe average Turn around time : %f ",avg_tat);
    printf("\nThe average Waiting time : %f ",avg_wt);
    printf("\nThe average response time : %f",avg_rt);
    printf("\nThe through put : %f ",t_put);
}


void main()
{
    int i;
    //clrscr();
    printf("CPU SCHEDULING ALGORITHMS\n");
    printf("(NON PEEMPTIVE MODE)\n");
    printf("1. First Come First Serve    2. Shortest Job First    3. Priority\n");
    printf("(PREEMPTIVE MODE)\n");
    printf("4. Round Robin\n");
    while(1)
    {
        printf("Enter your choice : ");
        scanf("%d",&ch);
        if(ch >= 1 && ch <= 6)
            break;
        else
            printf("Invlid choice choose again!!!\n");
    }
```

```
    switch (ch)
    {

        case 1 :    fcfs_data();
                    P_display();
                    fcfs_Sort();
                    fcfs_computations();
                    P_display();
                    final_res();
                    break;

        case 2 :    sjf_data();
                    P_display();
                    sjf_sort();
                    sjf_computations();
                    P_display();
                    final_res();
                    break;

        case 3 :    priority_data();
                    P_display();
                    priority_sort();
                    priority_computations();
                    P_display();
                    final_res();
                    break;
        case 4 :    RR_data();
                    RR_computations();
                    break;

    }
    //getch();
}
```

OUTPUT :

Test case : 1

```
CPU SCHEDULING ALGORITHMS
(NON PEEMPTIVE MODE)
1. First Come First Serve    2. Shortest Job First    3. Priority
(PREEMPTIVE MODE)
4. Round Robin
Enter your choice : 1
```

Enter the no. of processes : 5

Enter the process id's, arrival time and burst times of each process
1 0 4
2 1 3
3 2 1
4 3 2
5 4 5

prty	Pid	At	Bt	Ct	Tat	Wt	rt
0	1	0	4	0	0	0	0
0	2	1	3	0	0	0	0
0	3	2	1	0	0	0	0
0	4	3	2	0	0	0	0
0	5	4	5	0	0	0	0

prty	Pid	At	Bt	Ct	Tat	Wt	rt
0	1	0	4	4	4	0	0
0	2	1	3	7	6	3	3
0	3	2	1	8	6	5	5
0	4	3	2	10	7	5	5
0	5	4	5	15	11	6	6

The average Turn around time : 6.800000
The average Waiting time : 3.800000
The average response time : 3.800000
The through put : 0.333333

Test case : 2

CPU SCHEDULING ALGORITHMS
(NON PEEMPTIVE MODE)
1. First Come First Serve 2. Shortest Job First 3. Priority
(PREEMPTIVE MODE)
4. Round Robin
Enter your choice : 2
Enter the no. of processes : 5
Here we assume that all processes have arrived at 0
Enter the process id's and burst times of each process
1 7
2 5
3 1
4 2
5 8

prty	Pid	At	Bt	Ct	Tat	Wt	rt
0	1	0	7	0	0	0	0
0	2	0	5	0	0	0	0
0	3	0	1	0	0	0	0
0	4	0	2	0	0	0	0
0	5	0	8	0	0	0	0

prty	Pid	At	Bt	Ct	Tat	Wt	rt
0	3	0	1	1	1	0	0
0	4	0	2	3	3	1	1
0	2	0	5	8	8	3	3
0	1	0	7	15	15	8	8
0	5	0	8	23	23	15	15

The average Turn around time : 10.000000
The average Waiting time : 5.400000
The average response time : 5.400000
The through put : 0.217391

Test case : 3

CPU SCHEDULING ALGORITHMS
(NON PEEMPTIVE MODE)
1. First Come First Serve 2. Shortest Job First 3. Priority
(PREEMPTIVE MODE)
4. Round Robin
Enter your choice : 3
Enter the no. of processes : 7
Here we assume that all processes have arrived at 0
Lower the number higher the priority
Enter the priority, process id's and burst times of each process
2 1 4
4 2 2
6 3 3
10 4 5
8 5 1
12 6 4
9 7 6

prty	Pid	At	Bt	Ct	Tat	Wt	rt
2	1	0	4	0	0	0	0
4	2	0	2	0	0	0	0
6	3	0	3	0	0	0	0

```
10      4       0       5       0       0       0       0
8       5       0       1       0       0       0       0
12      6       0       4       0       0       0       0
9       7       0       6       0       0       0       0

prty    Pid     At      Bt      Ct      Tat     Wt      rt
2       1       0       4       4       4       0       0
4       2       0       2       6       6       4       4
6       3       0       3       9       9       6       6
8       5       0       1       10      10      9       9
9       7       0       6       16      16      10      10
10      4       0       5       21      21      16      16
12      6       0       4       25      25      21      21

The average Turn around time : 13.000000
The average Waiting time : 9.428572
The average response time : 9.428572
The through put : 0.280000
```

Test case : 4

```
CPU SCHEDULING ALGORITHMS
(NON PEEMPTIVE MODE)
1. First Come First Serve    2. Shortest Job First    3. Priority
(PREEMPTIVE MODE)
4. Round Robin
Enter your choice : 4
Enter the No. of processes : 6
 --------
| NOTE : | Here we are assuming all the processes have arrived at the Time stamp 0
 --------
Enter the burst times of the process
4  5  2  1  6  3
Enter Time Quantum : 4
Bt      tat     wt
4       4       0
5       19      14
2       10      8
1       11      10
6       21      15
3       18      15
Average Waiting Time: 13.833333
Average Turnaround Time: 10.333333
```

2. Simulate Bankers Algorithm for Dead Lock detection and deadlock Avoidance

```
#include<stdio.h>

int processes[20], max[20][20], allocated[20][20], need[20][20], resources[20],
available[20];
int n, r, process; // procces is for calling the safety algo for the specific task


void print_data()
{
    int i, j;
    printf("\nPrinting the data\n");
    printf("Process   Max     Allocated    Need\n");
    for ( i = 0; i < n; i++ )
    {
        printf("    %d     ", processes[i]);
        for ( j = 0; j < r; j++ )
        {
            printf("%d  ", max[i][j]);
        }
        printf("  ");
        for ( j = 0; j < r; j++ )
        {
            printf("%d  ", allocated[i][j]);
        }
        printf("  ");

        for ( j = 0; j < r; j++ )
        {
            printf("%d  ", need[i][j]);
        }
        printf("\n");
    }
    printf("The available resources \n");
    for ( i = 0; i < r; i++ )
        printf("%d  ",available[i]);
    printf("\n");
}
```

```
void release_algorithm( int process )
{
   int i, j, k = 0; // k will be the iterating variable for available array
   for ( i = 0; i < r; i++ )
   {
      available[k] += allocated[process][i];
      k++;
      allocated[process][i] = 0;
   }
   // updating the need matrix for corressponding procces
   for ( i = process, j = 0; j < r; j++ )
      need[process][j] = max[process][j];
}


void request_algorithm( int process )
{
   int i, j , k = 0, flag = 0, release_procees; // k will be the iterating variable
for available array
   for ( i = process, j = 0; j < r; j++ ) // Here r is No. of resources i.e to know
Howmany times we need to check for condition
   {
      if( need[i][j] > available[k++] )
      {
         flag = 1;
         break;
      }
   }

   if ( flag == 0)
   {
      printf("Access can be granted to procees %d \n", process);
      printf("Procces %d finished it execution \n", process);
      printf("Now the process %d releasing its resources \n", process);
      release_algorithm(process);
      for ( i = process, j =0; j < r; j++ )
      {
         max[process][j] = 0;
         need[process][j] = 0;
      }
   }
```

```
else
    {
        printf("Access denied to procees %d \n", process);
        printf("To execute the request procces %d, Enter a procces to release
it's resouces : ", process);
        scanf("%d", &release_procees);
        printf("On demand process %d releasing it's resources to avoid
Deadlock", release_procees);
        release_algorithm(release_procees);
        flag = 0, k = 0;
        for ( i = process, j = 0; j < r; j++ ) // Here r is No. of resources i.e to
know Howmany times we need to check for condition
        {
            if( need[i][j] > available[k++] )
            {
                flag = 1;
                break;
            }
        }
        print_data();
        if ( flag == 1 )
            printf("\nAfter procees %d release it's resouces process %d is unable
to execute", release_procees, process);
        else
        {
            printf("\nProcess %d finished it's execution", process);
            for ( i = process, j = 0; j < r; j++ )
                max[process][j] = 0;
            release_algorithm(process);
        }
    }
    print_data();
}


void main()
{
    int i, j, k = 0; // k is taken for storing the values in available array
    int sum = 0; // For calulating the sum of individual column sum of
allocated matrix to get available array values
    int process;
    printf("Enter the No. of procceses and resources : ");
    scanf("%d%d", &n, &r);
    for ( i = 0; i < n; i++ )
        processes[i] = i;
```

```
    printf("Enter the No. of resources of each type\n");
    for ( i = 0; i < r; i++ )
        scanf("%d",&resources[i]);
    printf("Enter the max resources needed for each process\n");
    for ( i = 0; i < n; i++ )
        for ( j = 0; j < r; j++ )
        {
            scanf("%d", &max[i][j]);
        }
    printf("Enter the allocated resources for each process\n");
    for ( i = 0; i < n; i++ )
        for ( j = 0; j < r; j++ )
        {
            scanf("%d", &allocated[i][j]);
        }

// calculating the need matrix
    for ( i = 0; i < n; i++ )
        for ( j = 0; j < r; j++ )
        {
            need[i][j] = max[i][j] - allocated[i][j];
        }

    for ( i = 0; i < r; i++ )
    {
        sum = 0;
        for ( j = 0; j < n; j++ )
            sum += allocated[j][i];
        available[k++] = resources[i] - sum;
    }
    // print them
    print_data();
    printf("Enter the procces to execute : ");
    scanf("%d", &process);
    request_algorithm(process);
}
```

OUTPUT

Test case : 1

```
Enter the No. of procceses and resources : 4  3
Enter the No. of resources of each type
10  8  6
```

```
Enter the max resources needed for each process
5  4  0
3  2  4
4  3  3
3  0  4
Enter the allocated resources for each process
2  3  0
2  0  3
1  3  2
2  0  1

Printing the data
Process   Max     Allocated    Need
  0    5  4  0    2  3  0    3  1  0
  1    3  2  4    2  0  3    1  2  1
  2    4  3  3    1  3  2    3  0  1
  3    3  0  4    2  0  1    1  0  3
The available resources
3  2  0
Enter the procces to execute : 0
Access can be granted to procees 0
Procces 0 finished it execution
Now the process 0 releasing its resources

Printing the data
Process   Max     Allocated    Need
  0    0  0  0    0  0  0    0  0  0
  1    3  2  4    2  0  3    1  2  1
  2    4  3  3    1  3  2    3  0  1
  3    3  0  4    2  0  1    1  0  3
The available resources
5  5  0
```

Test case : 1

```
Enter the No. of procceses and resources : 4  3
Enter the No. of resources of each type
10  8  6
Enter the max resources needed for each process
5  4  1
3  2  4
4  3  3
3  0  4
```

```
Enter the allocated resources for each process
2  3  0
2  0  3
1  3  2
2  0  1

Printing the data
Process   Max     Allocated    Need
  0    5  4  1    2  3  0    3  1  1
  1    3  2  4    2  0  3    1  2  1
  2    4  3  3    1  3  2    3  0  1
  3    3  0  4    2  0  1    1  0  3
The available resources
3  2  0
Enter the procces to execute : 0
Access denied to procees 0
To execute the request procces 0, Enter a procces to release it's resouces : 1
On demand process 1 releasing it's resources to avoid Deadlock
Printing the data
Process   Max     Allocated    Need
  0    5  4  1    2  3  0    3  1  1
  1    3  2  4    0  0  0    3  2  4
  2    4  3  3    1  3  2    3  0  1
  3    3  0  4    2  0  1    1  0  3
The available resources
5  2  3

Process 0 finished it's execution
Printing the data
Process   Max     Allocated    Need
  0    0  0  0    0  0  0    0  0  0
  1    3  2  4    0  0  0    3  2  4
  2    4  3  3    1  3  2    3  0  1
  3    3  0  4    2  0  1    1  0  3
The available resources
7  5  3
```

3. Simulate the Simulate paging technique

```
#include<stdio.h>

int mainmemory_size, max_processes, frame_size, max_pages,
remaining_pages;
int mm[100], pt[100], programs[20];

void paging()
{
   int i, j, index = 0, flag = 0; // index will be refering for the pagetable
   for ( i = 0; i < max_processes; i++ )
   {
      printf("No. of pages required for process[%d] : ", i);
      scanf("%d", &programs[i]);
      if ( programs[i] == 0 )
         continue;
      remaining_pages -= programs[i];
      if( programs[i] > max_pages )
      {
         printf("Requested memory is not available to fit the given pages in
the mainmemory\n");
         flag = 1;
         break;
      }
      else
      {
         if ( remaining_pages >= 0 )
         {
            printf("Enter the page table for process[%d] i.e (for %d pages) \n",
i, programs[i]);
            for ( j = 0; j < programs[i]; j++ )
               scanf("%d", &pt[index++]);
         }
         else
         {
            printf("Mainmemory is full\n");
            flag = 1;
            break;
         }
      }
   }
   if ( flag == 0)
      printf("Paging is implemented successfully\n");
```

```
    else
        printf("Paging is not implemented acccroding to the user
requirements\n");
}


void main()
{
    int i, j;
    printf("Enter the Main memory size (in bytes): ");
    scanf("%d", &mainmemory_size);
    printf("Enter the Frame or page size : ");
    scanf("%d", &frame_size);
    max_pages = mainmemory_size / frame_size;
    remaining_pages = max_pages;
    printf("Total no. of page available in main memory : %d\n", max_pages);
    printf("Enter the no. of procceses : ");
    scanf("%d", &max_processes);
    paging();
}
```

OUTPUT

Test case : 1

```
Enter the Main memory size (in bytes): 60
Enter the Frame or page size : 10
Total no. of page available in main memory : 6
Enter the no. of procceses : 4
No. of pages required for process[0] : 3
Enter the page table for process[0] i.e (for 3 pages)
10
20
30
No. of pages required for process[1] : 2
Enter the page table for process[1] i.e (for 2 pages)
35
40
No. of pages required for process[2] : 1
Enter the page table for process[2] i.e (for 1 pages)
50
No. of pages required for process[3] : 4
Mainmemory is full
Paging is not implemented acccroding to the user requirements
```

Test case : 2

```
Enter the Main memory size (in bytes): 60
Enter the Frame or page size : 10
Total no. of page available in main memory : 6
Enter the no. of procceses : 3
No. of pages required for process[0] : 2
Enter the page table for process[0] i.e (for 2 pages)
10
17
No. of pages required for process[1] : 1
Enter the page table for process[1] i.e (for 1 pages)
20
No. of pages required for process[2] : 3
Enter the page table for process[2] i.e (for 3 pages)
25
35
40
Paging is implemented successfully
```

4. Simulate page replacement algorithms
a) LRU b) Optimal

a. Least recently used page page replacement algorithm

```
#include<stdio.h>
// Remove comments if you execute in Turbo C compiler
//#include<conio.h>
int m_size, n, pf = 0;
int mm[20], search_array[25], values [20];
float phr,pfr;


int pagehit(int key)
{
    int i;
    for ( i = 0; i < m_size; i++ )
    {
        if ( mm[i] == key )
            return 1;
    }
    return 0;
}


int pagehitvalues(int key)
{
    int i;
    for ( i = 0; i < m_size; i++ )
    {
        if ( values[i] == key )
            return 1;
    }
    return 0;
}


void valuesIntialize()
{
    int i;
    for ( i = 0; i < m_size; i++ )
        values[i] = -1;
}
```

```
int lru_index(int lastindex)
{
    int i, j, k = 0, index = 0, data;
    valuesIntialize();
    for ( i = lastindex - 1; i >= 0; i-- )
    {
        if ( pagehit(search_array[i]) )
        {
            if( pagehitvalues(search_array[i]) == 0)
            {
                values[k] = search_array[i];
                k++;
            }
        }
    }

    index = m_size-1;
    data = values[index];
    for( i = 0; i < m_size; i++ )
    {
        if( mm[i] == data )
        {
            index = i;
            break;
        }
    }
    return index;
}


void page_replace(int index , int value)
{
    mm[index] = value;
}


void displayData()
{
    int i;
    for ( i = 0; i < m_size; i++ )
        printf("%d    ", mm[i]);
    printf("\n");
}
```

```
int main()
{
    int i, j = 0, index;
    //clrscr();
    printf("Enter the main memory size : ");
    scanf("%d", &m_size);
    printf("Enter the search array size : ");
    scanf("%d", &n);
    printf("Enter the search array : ");
    for ( i = 0; i < n; i++ )
        scanf("%d", &search_array[i]);
    for ( i = 0; i < m_size; i++ )
    {
        mm[i] = -1;
        values [i] = -1;
    }
    printf("Intially Manimemory is empty\n");
    printf("Printing the intial data in memory\n");
    displayData();
    printf("Printing the Mainmemory data during the LRU page replace
process\n");
    for ( i = 0; i < n; i++ )
    {
        if ( pagehit(search_array[i]) == 0)
        {
            pf++;
            if ( j < m_size )
                mm[j++] = search_array[i];
            else
            {
                index = lru_index(i);
                page_replace(index, search_array[i]);
            }
            displayData();
        }
    }

    printf("\nNo. of page faults : %d \n", pf);
    if ( pf == n )
    {
        pfr = 1;
        phr = 0;
    }
```

```
    else
    {
        phr = (float) (n-pf)/n;
        pfr = (float) pf/n;
    }
    printf("page hit ratio : %f \n", phr);
    printf("page fault ratio : %f \n", pfr);
    //getch();
    return 0;
}
```

OUTPUT

Test case : 1

```
Enter the main memory size : 3
Enter the search array size : 20
Enter the search array :
7  0  1  2  0  3  0  4  2  3
0  3  2  1  2  0  1  7  0  1
Intially Manimemory is empty
Printing the intial data in memory
-1    -1    -1
Printing the Mainmemory data during the LRU page replace process
7    -1    -1
7    0    -1
7    0    1
2    0    1
2    0    3
4    0    3
4    0    2
4    3    2
0    3    2
1    3    2
1    0    2
1    0    7

No. of page faults : 12
page hit ratio : 0.400000
page fault ratio : 0.600000
```

Test case : 2

```
Enter the main memory size : 4
Enter the search array size : 20
Enter the search array :
7  0  1  2  0  3  0  4  2  3
0  3  2  1  2  0  1  7  0  1
Intially Manimemory is empty
Printing the intial data in memory
-1    -1    -1    -1
Printing the Mainmemory data during the LRU page replace process
7    -1    -1    -1
7    0    -1    -1
7    0    1    -1
7    0    1    2
3    0    1    2
3    0    4    2
3    0    1    2
7    0    1    2

No. of page faults : 8
page hit ratio : 0.600000
page fault ratio : 0.400000
```

Test case : 3

```
Enter the main memory size : 3
Enter the search array size : 12
Enter the search array :
2  3  2  1  5  2  4  5  3  2  5  2
Intially Manimemory is empty
Printing the intial data in memory
-1    -1    -1
Printing the Mainmemory data during the LRU page replace process
2    -1    -1
2    3    -1
2    3    1
2    5    1
2    5    4
3    5    4
3    5    2
```

No. of page faults : 7
page hit ratio : 0.416667
page fault ratio : 0.583333

Test case : 4

Enter the main memory size : 3
Enter the search array size : 6
Enter the search array :
1 2 3 4 5 6
Intially Manimemory is empty
Printing the intial data in memory
-1 -1 -1
Printing the Mainmemory data during the LRU page replace process
1 -1 -1
1 2 -1
1 2 3
4 2 3
4 5 3
4 5 6

No. of page faults : 6
page hit ratio : 0.000000
page fault ratio : 1.000000

b. Optimal page page replacement algorithm

```
#include<stdio.h>
//#include<conio.h>

int m_size, n, pf = 0;
int mm[10], search_array[25], Index_arr[10]; // Index_arr array stores the
Index the value that not have been used for longer time
float phr,pfr;

int pagehit(int key)
{
   int i;
   for ( i = 0; i < m_size; i++ )
   {
      if ( mm[i] == key )
         return 1;
   }
   return 0;
}

void intializeIndex()
{
   int i;
   for ( i = 0; i < m_size; i++ )
      Index_arr[i] = 999;
}

int opt_replace(int startIndex)
{
   int i, j = 0, k = 0, index; // we didn't know which index to be replaced
   intializeIndex();
   for ( k = 0; k < m_size; k++ )
   {
      for( i = startIndex+1; i < n; i++ )
      {
         if(mm[j] ==  search_array[i] )
         {
            Index_arr[j] = i;
            j++;
            break;
         }
      }
   }
```

```
    index = 0; // assuming that 0th index page is not used for longest time
period
    for ( i = 0; i < m_size; i++ )
    {
        if(Index_arr[i] > Index_arr[index])
        {
            index = i;
        }
    }
    return index;
}


void page_replace(int index , int value)
{
    mm[index] = value;
}


void displayData()
{
    int i;
    for ( i = 0; i < m_size; i++ )
        printf("%d     ", mm[i]);
    printf("\n");
}


int main()
{
    int i, j = 0, index;
    //clrscr();
    printf("Enter the main memory size : ");
    scanf("%d", &m_size);
    printf("Enter the search array size : ");
    scanf("%d", &n);
    printf("Enter the search array : ");
    for ( i = 0; i < n; i++ )
        scanf("%d", &search_array[i]);
    for ( i = 0; i < m_size; i++ )
        mm[i] = -1;

    printf("Intially Manimemory is empty\n");
    printf("Printing the intial data in memory\n");
    displayData();
```

```
    printf("Printing the Mainmemory data during the LRU page replace
process\n");

    for ( i = 0; i < n; i++ )
    {
        if( pagehit(search_array[i]) == 0 )
        {
            pf++;
            if( j < m_size )
                mm[j++] = search_array[i];
            else
            {
                index = opt_replace(i);
                page_replace(index, search_array[i]);
            }
            displayData();
        }
    }

    printf("\nNo. of page faults : %d \n", pf);
    if ( pf == n )
    {
        pfr = 1;
        phr = 0;
    }
    else
    {
        phr = (float) (n-pf)/n;
        pfr = (float) pf/n;
    }
    printf("page hit ratio : %f \n", phr);
    printf("page fault ratio : %f \n", pfr);
    //getch();
    return 0;
}
```

OUTPUT

Test case : 1

```
Enter the main memory size : 3
Enter the search array size : 20
Enter the search array :
7  0  1  2  0  3  0  4  2  3  0  3  2  1  2  0  1  7  0  1
```

```
Intially Manimemory is empty
Printing the intial data in memory
-1    -1    -1
Printing the Mainmemory data during the LRU page replace process
7    -1    -1
7    0    -1
7    0    1
2    0    1
2    0    3
2    4    3
2    0    3
2    0    1
7    0    1

No. of page faults : 9
page hit ratio : 0.550000
page fault ratio : 0.450000
```

Test case : 2

```
Enter the main memory size : 4
Enter the search array size : 20
Enter the search array :
7  0  1  2  0  3  0  4  2  3  0  3  2  1  2  0  1  7  0  1
Intially Manimemory is empty
Printing the intial data in memory
-1    -1    -1    -1
Printing the Mainmemory data during the LRU page replace process
7    -1    -1    -1
7    0    -1    -1
7    0    1    -1
7    0    1    2
3    0    1    2
3    0    4    2
1    0    4    2
1    0    7    2

No. of page faults : 8
page hit ratio : 0.600000
page fault ratio : 0.400000
```

Test case : 3

```
Enter the main memory size : 3
Enter the search array size : 12
Enter the search array :
2  3  2  1  5  2  4  5  3  2  5  2
Intially Manimemory is empty
Printing the intial data in memory
-1    -1    -1
Printing the Mainmemory data during the LRU page replace process
2    -1    -1
2    3    -1
2    3    1
2    3    5
4    3    5
2    3    5

No. of page faults : 6
page hit ratio : 0.500000
page fault ratio : 0.500000
```

Test case : 4

```
Enter the main memory size : 4
Enter the search array size : 12
Enter the search array :
2  3  2  1  5  2  4  5  3  2  5  2
Intially Manimemory is empty
Printing the intial data in memory
-1    -1    -1    -1
Printing the Mainmemory data during the LRU page replace process
2    -1    -1    -1
2    3    -1    -1
2    3    1    -1
2    3    1    5
2    3    4    5

No. of page faults : 5
page hit ratio : 0.583333
page fault ratio : 0.416667
```

5. Simulate sequential and linked file allocation strategies

a. sequential

```
#include<stdio.h>
//#include<conio.h>

int harddisk[100], last[100], length[100], alloc[100];
int size, tot_blk_num, max_size;

void main()
{
   int i, j = 0, index = 0;
   //clrscr();
   printf("Enter the hard disk size : ");
   scanf("%d", &size);
   printf("Enter the Total no. of  blocks in memory : ");
   scanf("%d", &tot_blk_num);
   max_size = size/tot_blk_num;
   printf("Each block in memory can have store upto 5 kb max\n");
   // For the above line... You can take input from user and make necessary
changes in the code
   printf("Size of each file can have upto %d kb \n", max_size*5);
   // While giving file size as input give in multiples of 5..
   // Bcz we are working with Integers (ceil function is not applied here)
   printf("Enter the File sizes to be stored in Harddisk \n");
   for ( i = 0; i < tot_blk_num; i++ )
   {
      alloc[i] = -1;
      printf("Enter the file %d size : ",i);
      scanf("%d", &harddisk[i]);
      if( harddisk[i] > (max_size * 5) )
      {
         printf("File %d is not allocated in memory\n", i);
         harddisk[index] = 0;
      }
      else
      {
         printf("File %d is allocated in memory\n",i);
         index = index + ((harddisk[i])/5); // used in finding the length of file
         alloc[i] = i;
         last[j++] = index-1;
      }
   }
```

```
    for ( i = 0; i < tot_blk_num; i++ )
    {
        if ( i == 0)
            length[i] = last[i] + 1;// using 0 based indexing
        else
        {
            if(last[i] == 0)
                break;
            else
                length[i] = last[i] - last[i-1];
        }
    }
    printf("\nPrinting the Files data\n");
    printf("File No    start    length\n");
    for ( i = 0, j = 0; i < tot_blk_num; i++ )
    {
        if ( alloc[i] == i )
        {
            printf("File %d    %d    %d\n",alloc[i], (j*tot_blk_num), length[j]);
            j++;
        }
    }
    //getch();
}
```

OUTPUT

Test case : 1

```
Enter the hard disk size : 50
Enter the Total no. of  blocks in memory : 5
Each block in memory can have store upto 5 kb max
Size of each file can have upto 50 kb
Enter the File sizes to be stored in Harddisk
Enter the file 0 size : 55
File 0 is not allocated in memory
Enter the file 1 size : 25
File 1 is allocated in memory
Enter the file 2 size : 35
File 2 is allocated in memory
Enter the file 3 size : 45
File 3 is allocated in memory
Enter the file 4 size : 51
File 4 is not allocated in memory
```

Printing the Files data

File No	start	length
File 1	0	5
File 2	5	7
File 3	10	9

Test case : 2

Enter the hard disk size : 200
Enter the Total no. of blocks in memory : 10
Each block in memory can have store upto 5 kb max
Size of each file can have upto 100 kb
Enter the File sizes to be stored in Harddisk
Enter the file 0 size : 100
File 0 is allocated in memory
Enter the file 1 size : 210
File 1 is not allocated in memory
Enter the file 2 size : 80
File 2 is allocated in memory
Enter the file 3 size : 80
File 3 is allocated in memory
Enter the file 4 size : 50
File 4 is allocated in memory
Enter the file 5 size : 40
File 5 is allocated in memory
Enter the file 6 size : 30
File 6 is allocated in memory
Enter the file 7 size : 15
File 7 is allocated in memory
Enter the file 8 size : 12
File 8 is allocated in memory
Enter the file 9 size : 512
File 9 is not allocated in memory

Printing the Files data

File No	start	length
File 0	0	20
File 2	10	16
File 3	20	16
File 4	30	10
File 5	40	8
File 6	50	6
File 7	60	3
File 8	70	2

b. linked

```
#include<stdio.h>
// Remove comments if you execute in Turbo C compiler
//#include<conio.h>

int harddisk[100];
int harddisk_size, tot_size, tot_prt, blk_size , file_size;

void main()
{
   int i, ch;
   //clrscr();
   printf("Enter the hard disk size : ");
   scanf("%d", &harddisk_size);
   printf("Enter the no. of  partions in memory : ");
   scanf("%d", &tot_prt);
   printf("Enter each partion in memory can store upto max in (KB) : ");
   scanf("%d", &blk_size);
   tot_size = harddisk_size * blk_size;
   printf("%d\n", tot_size);
   while(1)
   {
      printf("Do you want to store file : (Any number for Yes and 0 for No) : ");
      scanf("%d", &ch);
      if ( ch == 0 )
         break;
      else
      {
         printf("Enter the File size to be stored in Hard disk : ");
         scanf("%d", &file_size);
         if(file_size <= tot_size)
         {
            printf("File can be stored\n");
            tot_size -=  file_size;
         }
         else
            printf("File cannot be stored\n");
      }
   }
   printf("Total available space in memory after storing the Files : %d KB",
tot_size);
   //getch();
}
```

OUTPUT

Test case : 1

```
Enter the hard disk size : 100
Enter the no. of  partions in memory : 10
Enter each partion in memory can store upto max in (KB) : 50
5000
Do you want to store file : (Any number for Yes and 0 for No) : 1
Enter the File size to be stored in Hard disk : 2048
File can be stored
Do you want to store file : (Any number for Yes and 0 for No) : -1
Enter the File size to be stored in Hard disk : 2048
File can be stored
Do you want to store file : (Any number for Yes and 0 for No) : 0
Total available space in memory after storing the Files : 904 KB
```

Test case : 2

```
Enter the hard disk size : 20
Enter the no. of  partions in memory : 5
Enter each partion in memory can store upto max in (KB) : 100
2000
Do you want to store file : (Any number for Yes and 0 for No) : 1
Enter the File size to be stored in Hard disk : 1024
File can be stored
Do you want to store file : (Any number for Yes and 0 for No) : 1
Enter the File size to be stored in Hard disk : 1024
File cannot be stored
Do you want to store file : (Any number for Yes and 0 for No) : 1
Enter the File size to be stored in Hard disk : 512
File can be stored
Do you want to store file : (Any number for Yes and 0 for No) : 1
Enter the File size to be stored in Hard disk : 128
File can be stored
Do you want to store file : (Any number for Yes and 0 for No) : 1
Enter the File size to be stored in Hard disk : 50
File can be stored
Do you want to store file : (Any number for Yes and 0 for No) : 0
Total available space in memory after storing the Files : 286 KB
```