ASSESSMENT - 8
SIDDHI SINGH
17BIT0028

QUES 1. Consider a memory hole of size 1kb initially. When a sequence of
memory request arrives as following, illustrate the memory allocation
by various approaches and calculate the total amount memory wasted
by external fragmentation and internal fragmentation in each approach.
a. First fit;
b. Best fit
c. Worst fit (Easy)

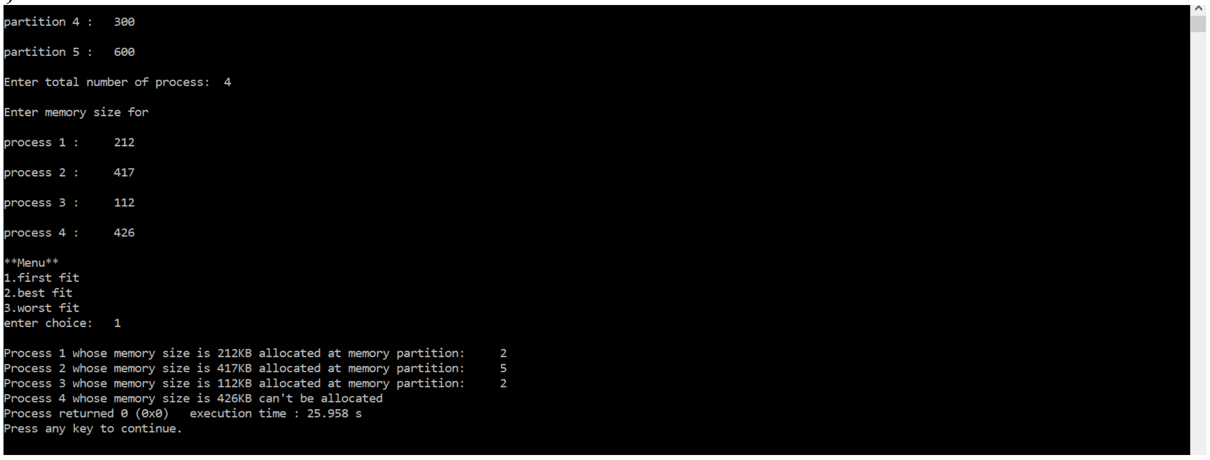
CODE

```
#include <iostream>
using namespace std;
int main()
{
  int c,i,j,k,n,l,m[10],p[10],po[20],flag,z,y,temp,temp1;
   cout<<"\t\t-----Welcome---"<<endl;
      cout<<"Enter memory total partitions:\t";
      cin>>n;
      cout<<"\nEnter memory size for\n";
      for(i=1;i<=n;i++)
      {
        cout<<"\npartition "<<i<<" :\t";
        cin>>m[i];
        po[i]=i;
      }
      cout<<"\nEnter total number of process:\t";
      cin>>j;
      cout<<"\nEnter memory size for\n";
      for(i=1;i<=j;i++)
      {
      cout<<"\nprocess "<<i<<" :\t";
        cin>>p[i];
      }
      cout<<"\n**Menu**\n1.first fit\n2.best fit\n3.worst fit\nenter choice:\t";
```

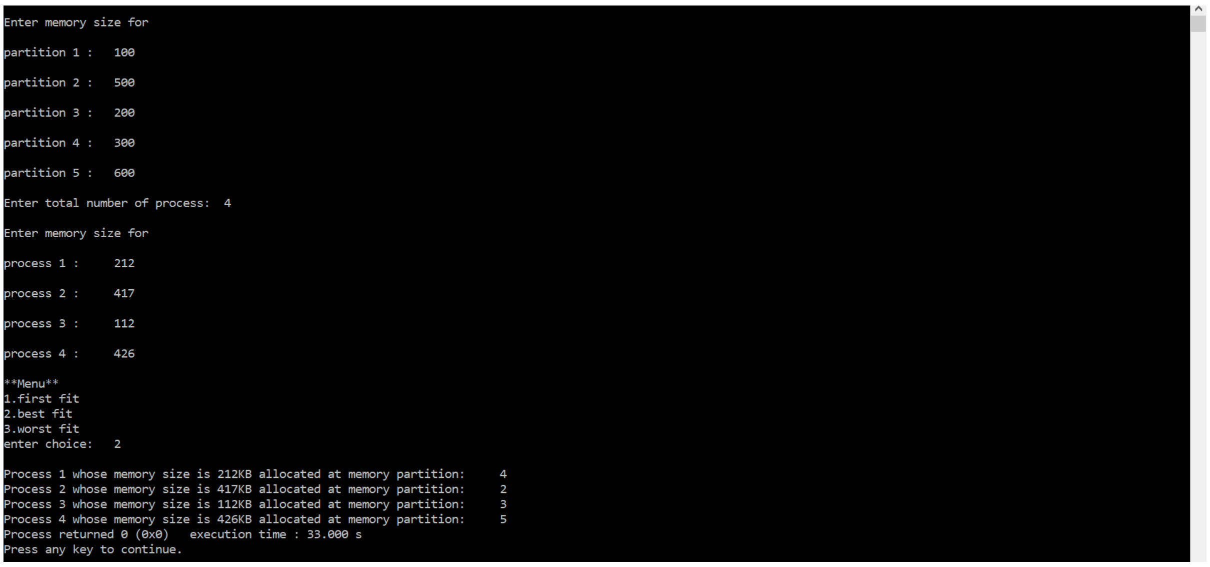
```
    cin>>c;
    switch(c)
    {
    case 1:
         for(i=1;i<=j;i++)
    {
       flag=1;
       for(k=1;k<=n;k++)
    {
       if(p[i]<=m[k])
       {
         cout<<"\nProcess "<<i<<" whose memory size is "<<p[i]<<"KB allocated at
memory partition:\t"<<po[k];
         m[k]=m[k]-p[i];
         break;
       }
       else
      {
        flag++;
       }
    }
    if(flag>n)
    {
      cout<<"\nProcess "<<i<<" whose memory size is "<<p[i]<<"KB can't be allocated";
    }
    }
    break;
    case 2:
     for(y=1;y<=n;y++)
       {
       for(z=y;z<=n;z++)
       {
       if(m[y]>m[z])
       {
       temp=m[y];
       m[y]=m[z];
       m[z]=temp;
       temp1=po[y];
       po[y]=po[z];
       po[z]=temp1;
       }
       }
       }
       for(i=1;i<=j;i++)
    {
       flag=1;
```

```
        for(k=1;k<=n;k++)
     {
        if(p[i]<=m[k])
        {
          cout<<"\nProcess "<<i<<" whose memory size is "<<p[i]<<"KB allocated at
memory partition:\t"<<po[k];
          m[k]=m[k]-p[i];
          break;
        }
        else
       {
         flag++;
        }
     }
     if(flag>n)
     {
       cout<<"\nProcess "<<i<<" whose memory size is "<<p[i]<<"KB can't be allocated";
     }
     }
        break;
        case 3:
        for(y=1;y<=n;y++)
        {
        for(z=y;z<=n;z++)
        {
        if(m[y]<m[z])
        {
        temp=m[y];
        m[y]=m[z];
        m[z]=temp;
        temp1=po[y];
        po[y]=po[z];
        po[z]=temp1;
        }
        }
        }
        for(i=1;i<=j;i++)
     {
        flag=1;
        for(k=1;k<=n;k++)
     {
        if(p[i]<=m[k])
        {
          cout<<"\nProcess "<<i<<" whose memory size is "<<p[i]<<"KB allocated at
memory partition:\t"<<po[k];
          m[k]=m[k]-p[i];
```

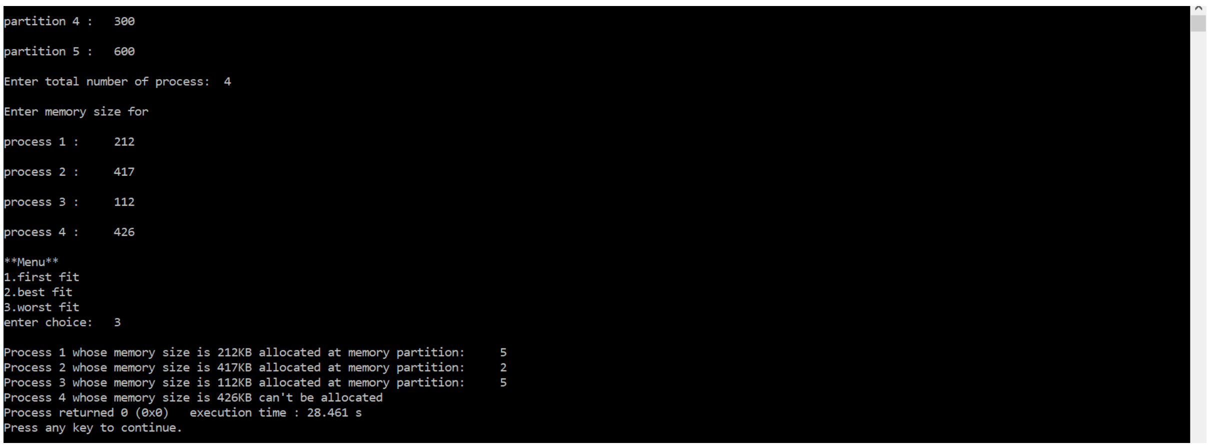
```
            break;
          }
          else
         {
           flag++;
          }
      }
      if(flag>n)
      {
        cout<<"\nProcess "<<i<<" whose memory size is "<<p[i]<<"KB can't be allocated";
      }
      }
          break;
          }
      return 0;
}
```



```
partition 4 :   300

partition 5 :   600

Enter total number of process:  4

Enter memory size for

process 1 :     212

process 2 :     417

process 3 :     112

process 4 :     426

**Menu**
1.first fit
2.best fit
3.worst fit
enter choice:   1

Process 1 whose memory size is 212KB allocated at memory partition:     2
Process 2 whose memory size is 417KB allocated at memory partition:     5
Process 3 whose memory size is 112KB allocated at memory partition:     2
Process 4 whose memory size is 426KB can't be allocated
Process returned 0 (0x0)   execution time : 25.958 s
Press any key to continue.
```



```
Enter memory size for

partition 1 :   100

partition 2 :   500

partition 3 :   200

partition 4 :   300

partition 5 :   600

Enter total number of process:  4

Enter memory size for

process 1 :     212

process 2 :     417

process 3 :     112

process 4 :     426

**Menu**
1.first fit
2.best fit
3.worst fit
enter choice:   2

Process 1 whose memory size is 212KB allocated at memory partition:     4
Process 2 whose memory size is 417KB allocated at memory partition:     2
Process 3 whose memory size is 112KB allocated at memory partition:     3
Process 4 whose memory size is 426KB allocated at memory partition:     5
Process returned 0 (0x0)   execution time : 33.000 s
Press any key to continue.
```



```
partition 4 :   300

partition 5 :   600

Enter total number of process:  4

Enter memory size for

process 1 :     212

process 2 :     417

process 3 :     112

process 4 :     426

**Menu**
1.first fit
2.best fit
3.worst fit
enter choice:   3

Process 1 whose memory size is 212KB allocated at memory partition:     5
Process 2 whose memory size is 417KB allocated at memory partition:     2
Process 3 whose memory size is 112KB allocated at memory partition:     5
Process 4 whose memory size is 426KB can't be allocated
Process returned 0 (0x0)   execution time : 28.461 s
Press any key to continue.
```

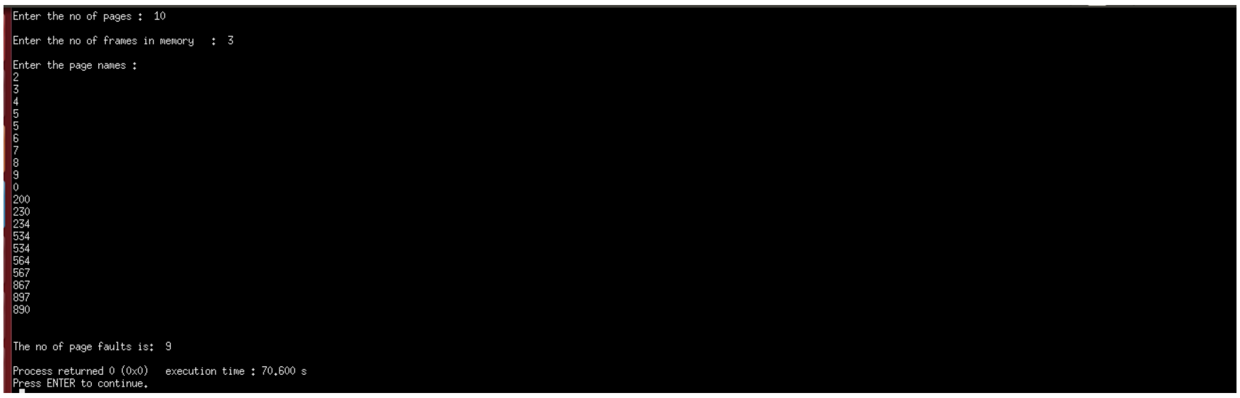
QUES 2. Write a program to implement the page replacement algorithms.
a. FIFO
b. LRU
c. OPT (Medium)

(a.)FIFO

<u>CODE</u>

```
#include<iostream>

using namespace std;

int main()
{
	int n,m,i,j,k,a,flag=0,pos,hits=0,flag1=0,pos1=0;
	int fr[30][10],p[30];
	cout<<"Enter the no of pages :  ";
	cin>>n;
	cout<<"\nEnter the no of frames in memory   :  ";
	cin>>m;
	cout<<"\nEnter the page names :\n";
	for(i=0;i<n;i++)
	{
		cin>>p[i];
	}
	//initialization
	for(i=0;i<n;i++)
```

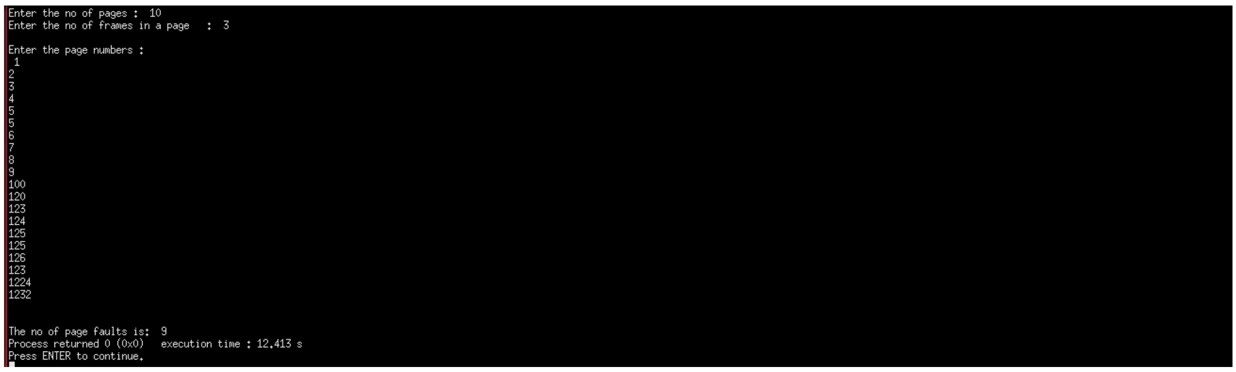
```
{
          for(j=0;j<m;j++)
          {  fr[i][j]=0;  }
}
//algorithm
for(i=0;i<n;i++)
{
          flag=0;flag1=0;
          if(i>0)
          {
                    a=i;
                    for(k=0;k<m;k++)
                    {
                                        fr[i][k]=fr[a-1][k];
                    }
          }
          for(j=0;j<m;j++)
          {
                    if(p[i]==fr[i][j])
                    {
                              flag=1;
                              hits++;
                    }
          }
          if(flag==0)
          {
                    for(k=0;k<m;k++)
                    {
                              if(fr[i][k]==0)
                              {
                                        pos=k;
                              flag1=1;
                              break;
                              }
                    }
                    if(flag1==1)
                    {
                              fr[i][pos]=p[i];
                    pos1=pos+1;
                    if(pos1>=m)
                                        pos1=0;
                    }
                    else
                    {
                              fr[i][pos1]=p[i];
                              pos1=pos1+1;
                    if(pos1>=m)
                            pos1=0;
                }
            }
            else continue;
        }
        //printing
        for(i=0;i<n;i++)
        {
            for(j=0;j<m;j++)
            {
                cout<<fr[i][j];
            }
            cout<<"\n";
        }
        cout<<"\n\nThe no of page faults is:  "<<n-hits<<endl;
        return 0;
}
```



```
Enter the no of pages :  10

Enter the no of frames in memory   :  3

Enter the page names :
2
3
4
5
5
6
7
8
9
0
200
230
234
534
534
564
567
867
897
890


The no of page faults is:  9

Process returned 0 (0x0)   execution time : 70.600 s
Press ENTER to continue.
```

OPT
CODE

```
//Optimal Page Replacement
#include<iostream>
//#include<conio.h>
using namespace std;

int main()
{
 int n,m,i,j,k,l=0,a,len,count=0,flag=0,pos=0,hits=0,flag1=0,pos1=0,pos2=0,max;
 int fr[30][10],p[30],temp[4];
 cout<<"Enter the no of pages :  "; cin>>n;
 cout<<"Enter the no of frames in a page   :  ";   cin>>m;
 cout<<"\nEnter the page numbers :\n ";
 for(i=0;i<n;i++)
 {
  cin>>p[i];
 }

//initialization

for(i=0;i<n;i++)
{  for(j=0;j<m;j++)
   {  fr[i][j]=0;  }
}

//algorithm

 for(i=0;i<n;i++)
 {
  flag=0;flag1=0;
  if(i>0)
  {
    a=i;
    for(k=0;k<m;k++)
    {
      fr[i][k]=fr[a-1][k];
    }
  }
  for(j=0;j<m;j++)
  {
    if(p[i]==fr[i][j])
    {    flag=1;  hits++; }

  }
  if(flag==0)
  {
    for(k=0;k<m;k++)
    {
      if(fr[i][k]==0)
      {   pos=k;
          flag1=1;
                   break;
      }
    }
    if(flag1==1)
    {  fr[i][pos]=p[i];
    }
    else
    {
      for(j=0;j<m;j++)
      {
        len=i+1;
        while(fr[i][j]!=p[len])
        { len++;count++; }
        temp[l++]=count+1;
        count=0;
      }
      max=temp[0];
      for(j=0;j<m;j++)
      {
        if(temp[j]>max) { max=temp[j]; pos2=j; }
      }
      fr[i][pos2]=p[i];
    }
  }
  else continue;
 }

//printing

for(i=0;i<n;i++)
{
 for(j=0;j<m;j++)
 {
  cout<<fr[i][j];
 }
 cout<<"\n";
}
cout<<"\n\nThe no of page faults is:  "<<n-hits;
//getch();
return 0;
}
```

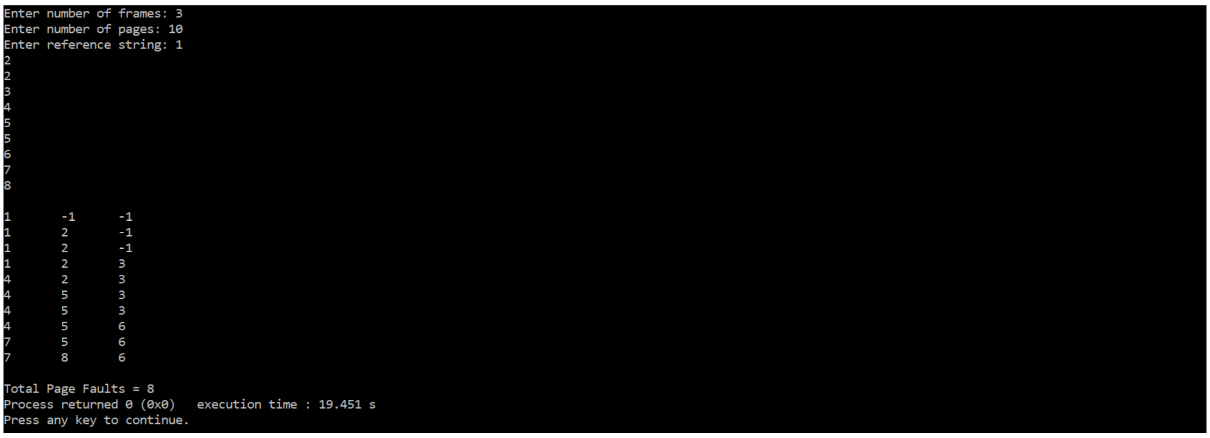


```
Enter the no of pages :  10
Enter the no of frames in a page   :  3

Enter the page numbers :
 1
2
3
4
5
5
6
7
8
9
100
120
123
124
125
125
126
123
1224
1232


The no of page faults is:  9
Process returned 0 (0x0)   execution time : 12.413 s
Press ENTER to continue.
```

LRU

CODE

```
#include<stdio.h>

int findLRU(int time[], int n){
   int i, minimum = time[0], pos = 0;

   for(i = 1; i < n; ++i){
      if(time[i] < minimum){
         minimum = time[i];
         pos = i;
      }
   }

   return pos;
}

int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2,
i, j, pos, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);

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

   printf("Enter reference string: ");

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

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

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

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

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

        if(flag2 == 0){
            pos = findLRU(time, no_of_frames);
            counter++;
            faults++;
            frames[pos] = pages[i];
            time[pos] = counter;
        }

        printf("\n");

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

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

    return 0;
}
```



```
Enter number of frames: 3
Enter number of pages: 10
Enter reference string: 1
2
2
3
4
5
5
6
7
8

1       -1      -1
1       2       -1
1       2       -1
1       2       3
4       2       3
4       5       3
4       5       3
4       5       6
7       5       6
7       8       6

Total Page Faults = 8
Process returned 0 (0x0)   execution time : 19.451 s
Press any key to continue.
```

QUES 3. Write a program that implements the FIFO, LRU, and optimal pager replacement algorithms. First, generate a random page- reference string where page numbers range from 0 to 9. Apply the random page reference string to each algorithm, and record the number of page faults incurred by each algorithm.Implement the replacement algorithms so that the number of page frames can vary from 1 to 7. Assume that demand paging is used

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


/**************************************
 * The parameters of memory and disk pages
 *
 * PageFrameList: The dynamically-allocated array representing
memory pages
 * FrameNR: the number of page frames in memory
 * elementCout: Point to the next page to be replaced
 *
 * ReferenceString: The sequence of the demanding pages
 * ReferenceLength: The length of the randomized reference
string
 * ReferenceSZ: the page number range in the reference string
 *
 */
```

```
#define ReferenceLength 100

typedef struct
{
    int *PageFrameList;
    int elementCount;
}PageFrame;

int ReferenceSZ, FrameNR;

PageFrame memory;

int *ReferenceString;


/* Test driver sub functions */

void generateReferenceString();

void initializePageFrame();

void printReferenceString();

void printPageFrame();


/* Algorithm Functions */

int FIFO();

int LRU();

/* The possible algorithm subfunctions

int FIFOSearch(int PageNumber);

int FIFOInsert(int PageNumber);

int LRUSearch(int PageNumber);

int LRUInsert(int PageNumber);

void LRUupdatePageTable(int Findindex);

*/



/*******************************
```

```
 *
 * The main function is the test driver for FIFO & LRU
algorithms
 *
 * 1. Initialize the system parameters
 * 2. Initialize the memory pages
 * 3. Generate the randomized reference string
 * 4. Apply the FIFO algorithm, calculate the number of page
faults
 * 5. Apply the LRU algorithm, calculate the number of page
faults
 */


int main(int argc, char* argv[])
{



    if( argc != 3 )
    {
        printf("Command format: Test <reference string size>
<number of page frames>");
    }


    ReferenceSZ = atoi(argv[1]);
    FrameNR = atoi(argv[2]);


   generateReferenceString();


   initializePageFrame();
   printf("page fault of FIFO: %d\n",FIFO());
   free(memory.PageFrameList);

   printf("\n");
   printf("\n");


   printReferenceString();

   initializePageFrame();
   printf("page fault of LRU: %d\n",LRU());
   free(memory.PageFrameList);


   free(ReferenceString);
   return 0;

}


/**********************************
 **********************************
 *
 * The test driver functions implmentation details
 *
 **********************************
 */

void generateReferenceString()
{
   int i;
   srand(time(0));
   ReferenceString = (int *)malloc( sizeof(int) *
ReferenceLength );
   printf("The randomized Reference String: ");
   for(i=0; i< ReferenceLength; i++)
   {
     ReferenceString[i] = rand() % ReferenceSZ;
        printf("%d ", ReferenceString[i]);
   }
   printf("\n");
}


void initializePageFrame()
{
   int i;
   memory.PageFrameList = (int *)malloc( sizeof(int)*
FrameNR );
   memory.elementCount =0;
   for(i=0; i< FrameNR; i++)
   {
     memory.PageFrameList[i] = -1;
   }

}

void printPageFrame()
{
   int i;
   for(i=0; i< FrameNR; i++)
   {
     printf("%2d ",memory.PageFrameList[i]);
   }
   printf("\n");
}

void printReferenceString()
{
   int i;
   printf("The Same Reference String: ");
   for(i=0; i< ReferenceLength; i++)
   {
        printf("%d ", ReferenceString[i]);
   }
   printf("\n");

}


/**********************************
 **********************************
 *
 * The skeleton code for FIFO & LRU algs
 *
 * NOTE: you are not required to follow the skeleton code here
 *       you can also write on your own, even different data
structure
 *       BUT make sure your algorithm is correct!!!!!!
 *       It is strongly recommended to print out the
PageFrames
 *       so that you can follow how the algorithm works and
double check it.
 *
 **********************************
 */


int FIFO()
{
    int PagefaultCount=0;
    int i;

   for( i=0; i<ReferenceLength; i++ )
   {
       PagefaultCount+=FIFOInsert(ReferenceString[i]);
       printPageFrame();
   }


   return PagefaultCount;
}
```

```
/*  Some hints you can follow
int FIFOSearch(int PageNumber)
{



}

int FIFOInsert(int PageNumber)
{
    int Pagefault=0;
    if( 0==FIFOSearch(PageNumber) )
    {

      //Replace the page HERE

    }

    return Pagefault;
}

*/


int LRU()
{
    int PagefaultCount=0;
    int i;

   for( i=0; i<ReferenceLength; i++ )
   {
       PagefaultCount+=LRUInsert(ReferenceString[i]);
       printPageFrame();
   }


   return PagefaultCount;

}

/*  Some hints you can follow
int LRUSearch(int PageNumber)
{


}
```

```
int LRUInsert(int PageNumber)
{
    int PageFault=0;

    int Findindex = -1;
    Findindex = LRUSearch(PageNumber);

    if ( -1 == Findindex )
    {

      //Replace the page HERE

    }
    else
    {
       LRUupdatePageTable(Findindex);
    }
    return PageFault;

}

void LRUupdatePageTable(int Findindex)
{



}
*/
```