Kulvir Singh
19BCE2074

CSE2005 – Operating Systems Lab Assessment 4

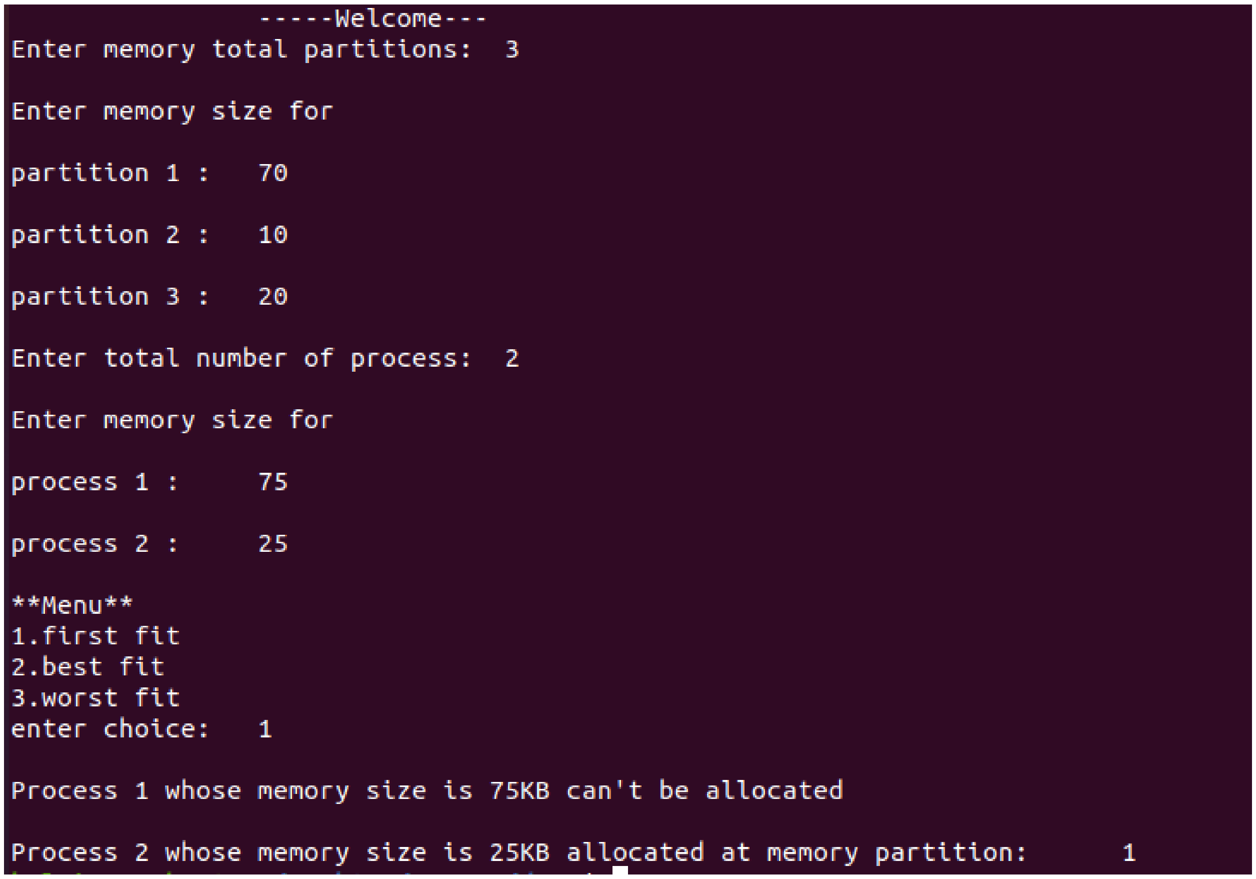
a)Consider a memory hole of size 1kb initially. When a sequence of memory request arrives as following, illustrate the memory allocation by various approaches and calculate the total amount memory wasted by external fragmentation and internal fragmentation in each approach.
i. First fit; ii. Best fit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

<u>CODE :</u>

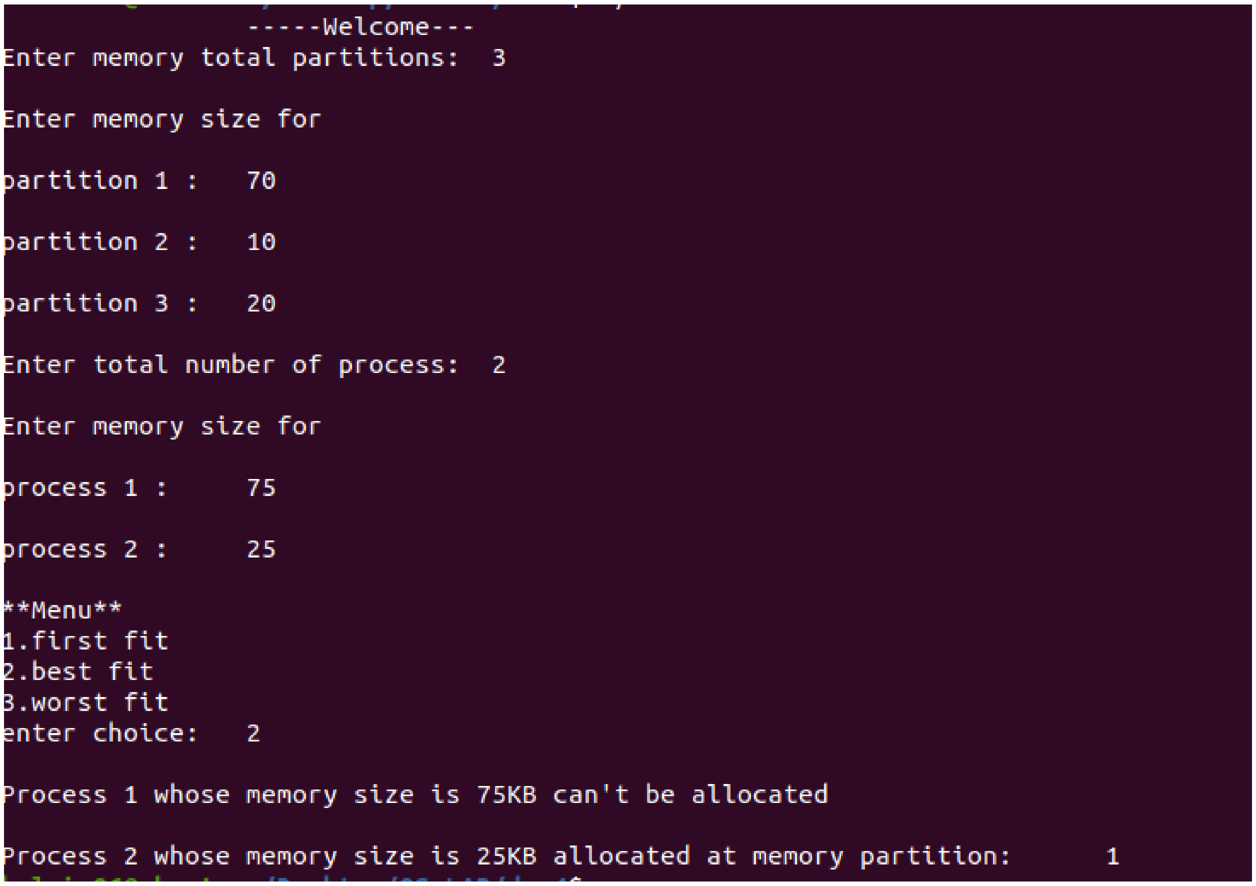
```
#include <iostream>
using namespace std;
int main()
{
 int c,i,j,k,n,l,m[10],p[10],po[20],flag,z,y,temp,temp1;
  cout<<"\t\t-----Welcome---"<<endl;
   cout<<"Enter memory total partitions:\t";
   cin>>n;
   cout<<"\nEnter memory size for\n";
   for(i=1;i<=n;i++)
   {
    cout<<"\npartition "<<i<<" :\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]<<"\n";
         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\n";
    }
    }
    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]<<"\n";
            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\n";
      }
      }
          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]<<"\n";
            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\n";
    }
    }
       break;
       }
    return 0;
}
```

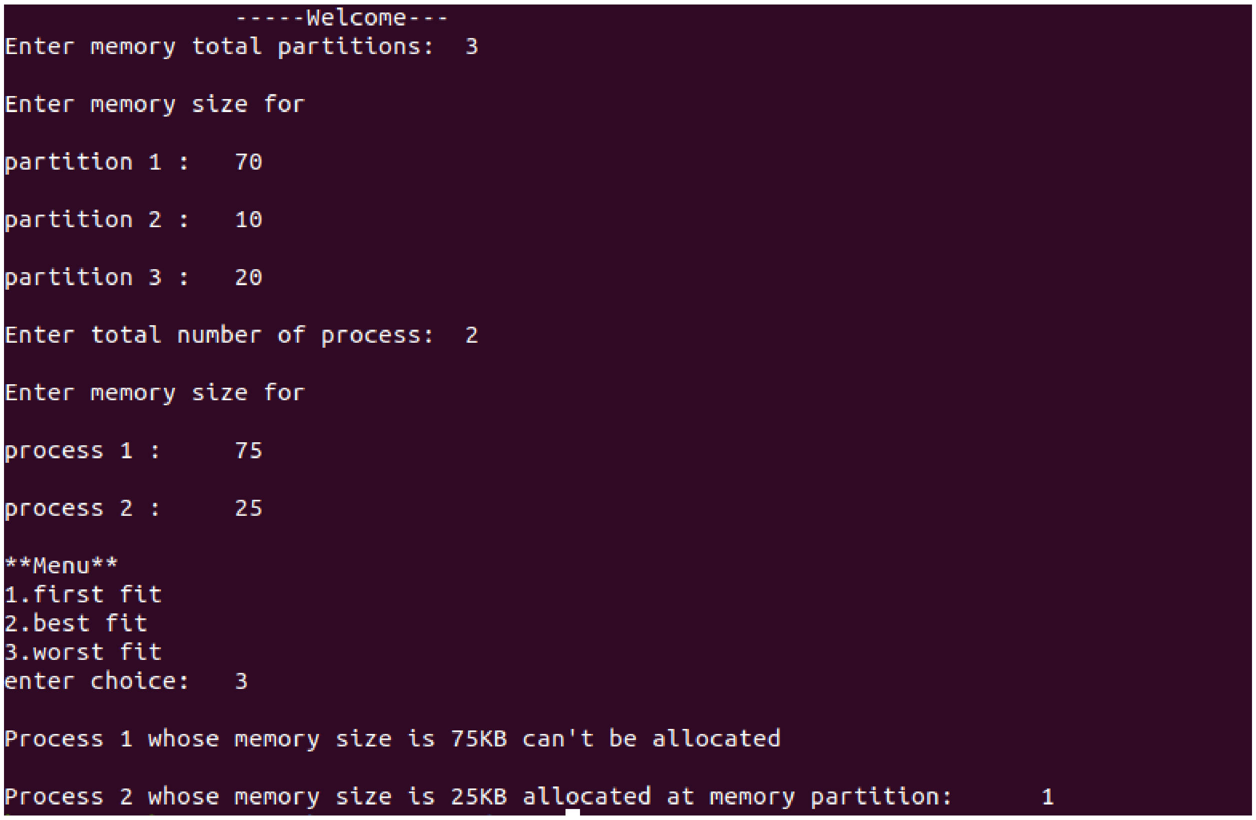
OUTPUT SCREENSHOTS :



```
                -----Welcome---
Enter memory total partitions:  3

Enter memory size for

partition 1 :   70

partition 2 :   10

partition 3 :   20

Enter total number of process:  2

Enter memory size for

process 1 :     75

process 2 :     25

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

Process 1 whose memory size is 75KB can't be allocated

Process 2 whose memory size is 25KB allocated at memory partition:      1
```



```
                -----Welcome---
Enter memory total partitions:  3

Enter memory size for

partition 1 :   70

partition 2 :   10

partition 3 :   20

Enter total number of process:  2

Enter memory size for

process 1 :     75

process 2 :     25

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

Process 1 whose memory size is 75KB can't be allocated

Process 2 whose memory size is 25KB allocated at memory partition:      1
```



```
                -----Welcome---
Enter memory total partitions:  3

Enter memory size for

partition 1 :   70

partition 2 :   10

partition 3 :   20

Enter total number of process:  2

Enter memory size for

process 1 :     75

process 2 :     25

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

Process 1 whose memory size is 75KB can't be allocated

Process 2 whose memory size is 25KB allocated at memory partition:      1
```

b)Write a program to implement the page replacement algorithms.
i. FIFO ii. LRU iii. OPT

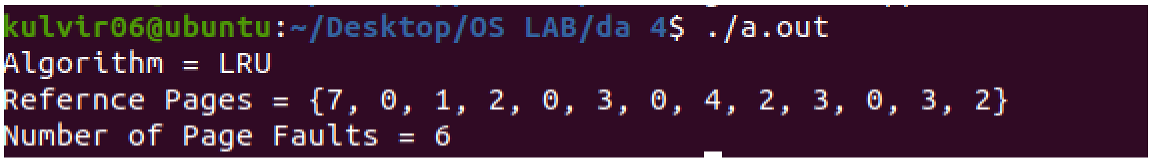
LRU:

CODE :

```
#include<bits/stdc++.h>
using namespace std;
int pageFaults(int pages[], int n, int capacity)
{
	unordered_set<int> s;
	unordered_map<int, int> indexes;
	int page_faults = 0;
	for (int i=0; i<n; i++)
	{
		if (s.size() < capacity)
		{

			if (s.find(pages[i])==s.end())
			{
				s.insert(pages[i]);

				page_faults++;
			}

			indexes[pages[i]] = i;
		}

		else
		{

			if (s.find(pages[i]) == s.end())
			{

				int lru = INT_MAX, val;
				for (auto it=s.begin(); it!=s.end(); it++)
				{
					if (indexes[*it] < lru)
					{
						lru = indexes[*it];
						val = *it;
					}
				}

				// Remove the indexes page
				s.erase(val);
```

```
                        // insert the current page
                        s.insert(pages[i]);

                        // Increment page faults
                        page_faults++;
                    }

                    // Update the current page index
                    indexes[pages[i]] = i;
                }
            }

            return page_faults;
}

// Driver code
int main()
{
            int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
            int n = sizeof(pages)/sizeof(pages[0]);
            int capacity = 4;
            cout<<"Algorithm = LRU\n";
            cout<<"Refernce Pages = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2}\n";
            cout <<"Number of Page Faults = " <<pageFaults(pages, n, capacity)<<"\n";
            return 0;
}
```

<u>OUTPUT SCREENSHOTS :</u>



```
kulvir06@ubuntu:~/Desktop/OS LAB/da 4$ ./a.out
Algorithm = LRU
Refernce Pages = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2}
Number of Page Faults = 6
```

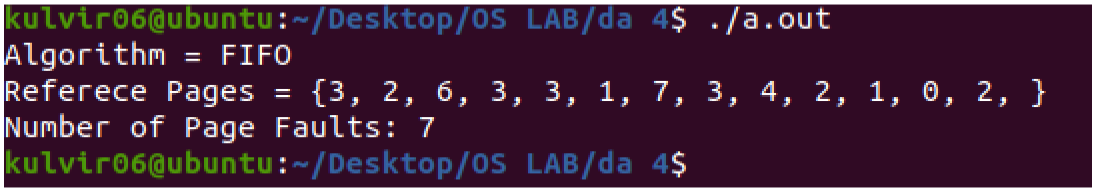
FIFO:

CODE :

```
#include <bits/stdc++.h>
#include <string>
#include <iostream>
using namespace std;
int pageFaults (int pages[], int n, int capacity)
{
   unordered_set <int> s;
   queue <int> indexes;
   int page_faults = 0;
   for(int i = 0; i < n; i++)
   {
      if(s.size () < capacity) {
         if(s.find (pages[i]) == s.end ()) {
            s.insert (pages[i]);
                  page_faults++;
                  indexes.push (pages[i]);
          }
     }
     else {
        if(s.find (pages[i]) == s.end ()) {
        int val = indexes.front ();
        indexes.pop ();
        s.erase (val);
        s.insert (pages[i]);
        indexes.push (pages[i]);
        page_faults++;
        }
     }
   }
   return page_faults;
}

// Driver code
int main() {
   int pages[] = { 3, 2, 6, 3, 3, 1, 7, 3, 4, 2, 1, 0, 2};
   int n = sizeof(pages) /   sizeof(pages[0]);
   int capacity = 7;
   cout << "Algorithm = FIFO \n";
   cout << "Referece Pages = ";
   cout << "{";
   for(int i = 0; i < n; i++) {
      cout << pages[i];  cout << ", ";
   }
   cout << "}";
```

```
    cout << "\nNumber of Page Faults: ";
    cout << pageFaults (pages, n, capacity);
    cout<<"\n";
    return 0;
}
```

OUTPUT SCREENSHOTS :



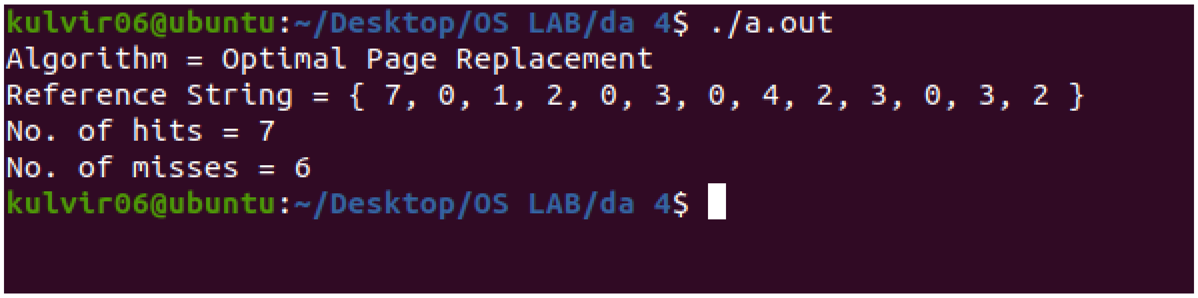
```
kulvir06@ubuntu:~/Desktop/OS LAB/da 4$ ./a.out
Algorithm = FIFO
Referece Pages = {3, 2, 6, 3, 3, 1, 7, 3, 4, 2, 1, 0, 2, }
Number of Page Faults: 7
kulvir06@ubuntu:~/Desktop/OS LAB/da 4$
```

OPT:

CODE :

```
#include <bits/stdc++.h>
using namespace std;

bool search(int key, vector<int>& fr)
{
        for (int i = 0; i < fr.size(); i++)
                if (fr[i] == key)
                        return true;
        return false;
}

int predict(int pg[], vector<int>& fr, int pn, int index)
{
        int res = -1, farthest = index;
        for (int i = 0; i < fr.size(); i++) {
                int j;
                for (j = index; j < pn; j++) {
                        if (fr[i] == pg[j]) {
                                if (j > farthest) {
                                        farthest = j;
                                        res = i;
                                }
                                break;
                        }
                }
        if (j == pn)
            return i;
    }
    return (res == -1) ? 0 : res;
}

void optimalPage(int pg[], int pn, int fn)
{
    vector<int> fr;
    int hit = 0;
    for (int i = 0; i < pn; i++) {
        if (search(pg[i], fr)) {
            hit++;
            continue;
        }
        if (fr.size() < fn)
            fr.push_back(pg[i]);
        else {
            int j = predict(pg, fr, pn, i + 1);
            fr[j] = pg[i];
        }
    }
    cout << "No. of hits = " << hit << endl;
    cout << "No. of misses = " << pn - hit << endl;
}
int main()
{
    int pg[] = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 };
    int pn = sizeof(pg) / sizeof(pg[0]);
    cout<<"Algorithm = Optimal Page Replacement\n";
    cout<<"Reference String = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 }\n";
    int fn = 4;
    optimalPage(pg, pn, fn);
    return 0;
}
```

<u>OUTPUT SCREENSHOTS :</u>



```
kulvir06@ubuntu:~/Desktop/OS LAB/da 4$ ./a.out
Algorithm = Optimal Page Replacement
Reference String = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 }
No. of hits = 7
No. of misses = 6
kulvir06@ubuntu:~/Desktop/OS LAB/da 4$
```

c)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.

CODE :

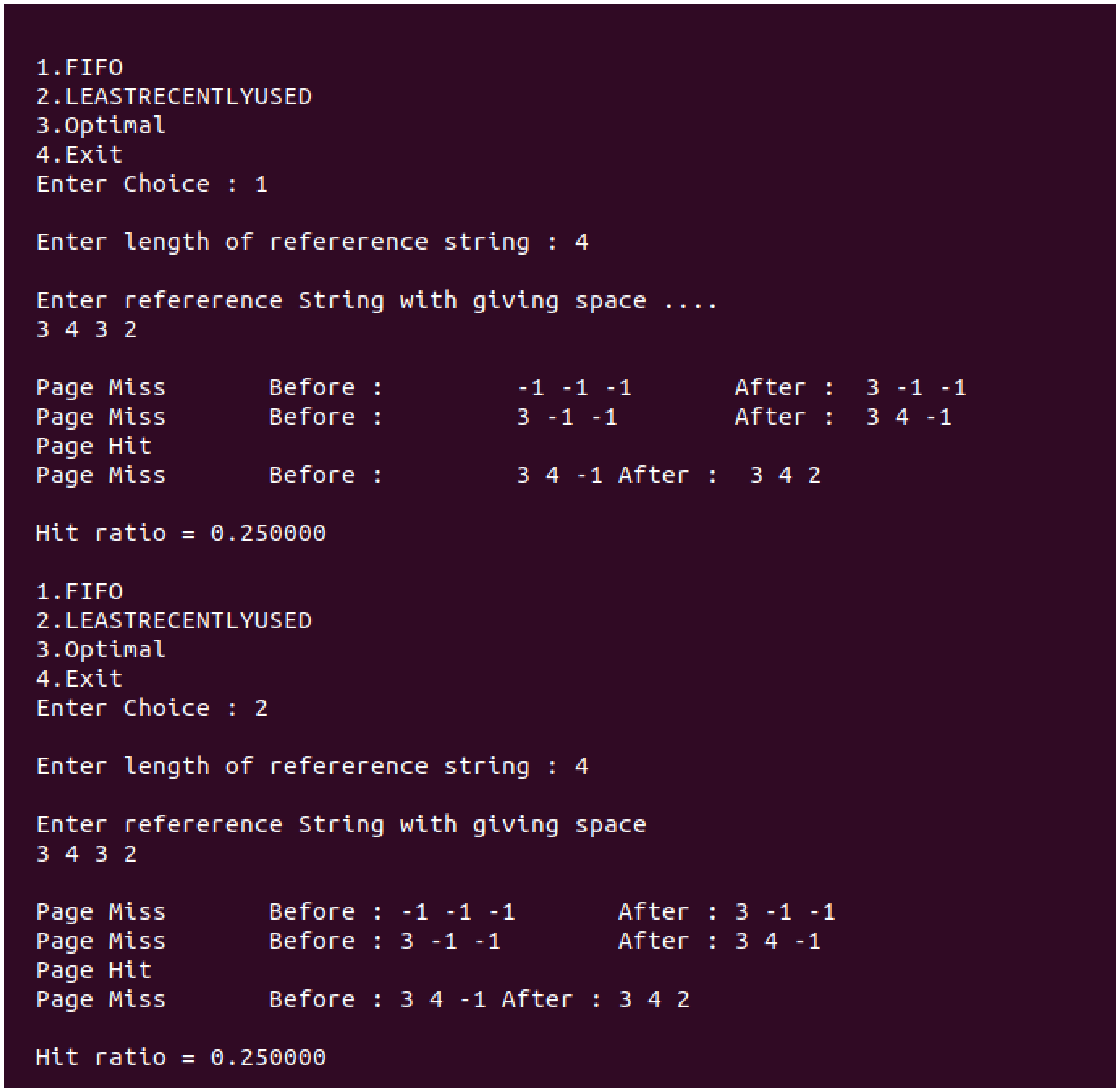
```
#include<stdio.h>
void FIFO();
void LEASTRECENTLYUSED();
void OPTIMAL();

int main()
{
   int ch;
   do
   {
      printf("\n\n\t1.FIFO\n\t2.LEASTRECENTLYUSED\n\t3.Optimal\n\t4.Exit\n\tEnter Choice : ");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1:
         FIFO();
         break;
         case 2:
         LEASTRECENTLYUSED();
         break;
         case 3:
         OPTIMAL();
         break;
      }
   }while(ch!=4);
}
void FIFO()
{
   int frame[3]={-1,-1,-1},refer[20],ctn=0,i,j,number,flag;
   float ratio,hitctn=0.00;
   printf("\n\tEnter length of refererence string : ");
   scanf("%d",&number);
   printf("\n\tEnter refererence String with giving space ....\n\t");
   for(i=0;i<number;i++)
   scanf("%d",&refer[i]);
```

```
    //printf("\n\tExecution is started here .....");
    for(i=0;i<number;i++)
    {
        flag=0;
        for(j=0;j<3;j++)
        if(frame[j]==refer[i])
        {
            printf("\n\tPage Hit ");
            hitctn++;
            flag=1;
            break;
        }

        if(flag==0)
        {
            printf("\n\tPage Miss");
            printf("\tBefore :\t");
            for(j=0;j<3;j++)
            printf(" %d",frame[j]);
            frame[ctn]=refer[i];
            ctn++;
            if(ctn>=3)
            ctn=0;
            printf("\tAfter :\t");
            for(j=0;j<3;j++)
            printf(" %d",frame[j]);
        }
    }
ratio=hitctn/number;
printf("\n\n\tHit ratio = %f ",ratio);
}
void LEASTRECENTLYUSED()
{
    int frame[3]={-1,-1,-1},used[3]={-1,-1,-1},ctn=0,refer[20],i,j,flag,number,index,value;
    float ratio,hitctn=0;
    printf("\n\tEnter length of refererence string : ");
    scanf("%d",&number);
    printf("\n\tEnter refererence String with giving space \n\t");
    for(i=0;i<number;i++)
    scanf("%d",&refer[i]);
    //printf("\n\tExecution is started ");
    for(i=0;i<number;i++)
    {
        flag=0;
        for(j=0;j<3;j++)
        if(frame[j]==refer[i])
        {
            printf("\n\tPage Hit ");
            hitctn++;
            flag=1;
            used[j]=ctn;
            break;
        }
        if(flag==0)
        {
            printf("\n\tPage Miss");
            printf("\tBefore :");
            for(j=0;j<3;j++)
            printf(" %d",frame[j]);
            //selection of Fault for replacement
            index=0;
            value=used[0];
            if(ctn!=0) {
            for(j=0;j<3;j++)
            if(value>used[j]&&value!=used[j])
            {
                index=j;
                value=used[j];
            }
        }
        //printf("\tFault is %d ",index);
        frame[index]=refer[i];
        used[index]=ctn;
        printf("\tAfter :");
        for(j=0;j<3;j++)
        printf(" %d",frame[j]);
    }
ctn++;
}
ratio=hitctn/number;
printf("\n\n\tHit ratio = %f ",ratio);
}
void OPTIMAL()
{
    int frame[3]={-1,-1,-1},used[3]={-1,-1,-1},ctn=0,refer[20],i,j,flag,number,value1,value2,value3,index;
    float ratio,hitctn=0;
    printf("\n\tEnter length of refererence string : ");
    scanf("%d",&number);
    printf("\n\tEnter refererence String with giving space \n\t");
    for(i=0;i<number;i++)
    scanf("%d",&refer[i]);
    //printf("\n\tExecution is started here ");
    for(i=0;i<number;i++)
    {
        flag=0;
        for(j=0;j<3;j++)
```

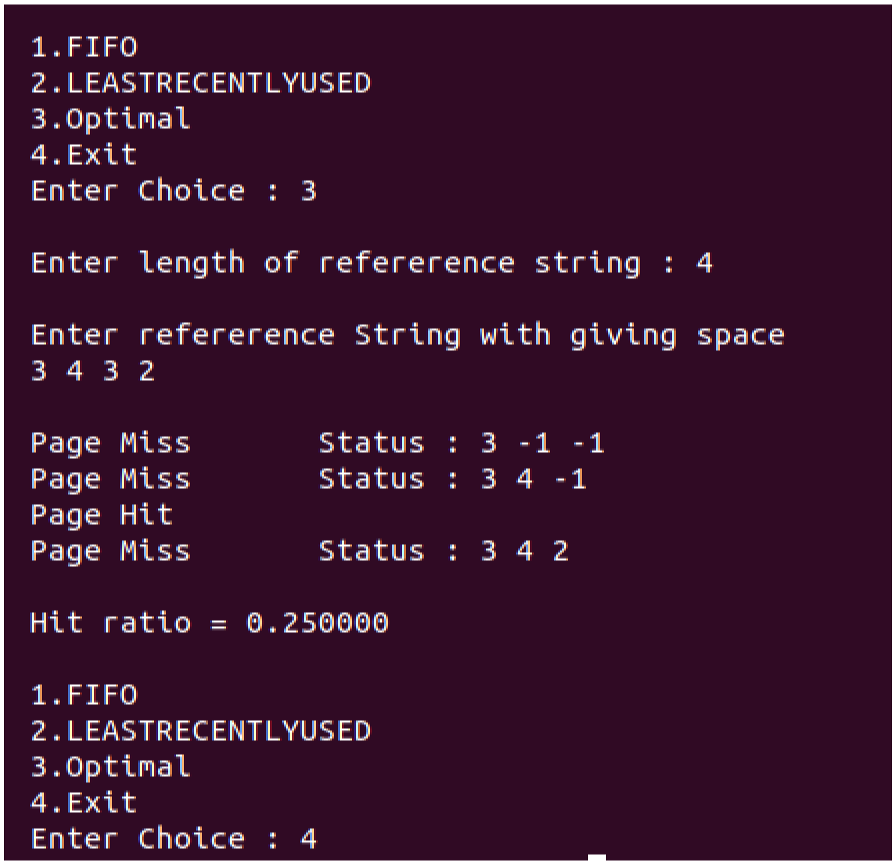
```
   if(frame[j]==refer[i])
   {
      flag=1;
      printf("\n\tPage Hit");
      hitctn++;
      break;
   }
if(flag==0)
{
printf("\n\tPage Miss");
if(ctn<3)
{
   frame[ctn]=refer[i];
   printf("\tStatus :");
   for(j=0;j<3;j++)
   printf(" %d",frame[j]);
   ctn++;
}
else
{
   printf("\tBefore :");
   for(j=0;j<3;j++)
   printf(" %d",frame[j]);
   //selection of Fault
   value1=frame[0];
   flag=0;
   for(j=i;j<number;j++)
   if(refer[j]==value1)
   {
      value1=j;
      flag=1;
      break;
   }
   if(flag==0)
   value1=number;
   value2=frame[1];
   flag=0;
   for(j=i;j<number;j++)
   if(refer[j]==value2)
   {
      value2=j;
      flag=1;
      break;
   }
   if(flag==0)
   value2=number;
   value3=frame[2];
   flag=0;
```

```
        for(j=i;j<number;j++)
        if(refer[j]==value3)
        {
           value3=j;
           flag=1;
           break;
        }
        if(flag==0)
           value3=number;
        if(value1<value2)
        if(value3<value2)
           index=1;
        else
           index=2;
        else
        if(value3<value1)
           index=0;
        else
           index=2;

        frame[index]=refer[i];
        printf("\tAfter :");
        for(j=0;j<3;j++)
           printf(" %d",frame[j]);
        }
     }
}
ratio=hitctn/number;
printf("\n\n\tHit ratio = %f ",ratio);
}
```

<u>OUTPUT SCREENSHOTS :</u>



```
1.FIFO
2.LEASTRECENTLYUSED
3.Optimal
4.Exit
Enter Choice : 1

Enter length of refererence string : 4

Enter refererence String with giving space ....
3 4 3 2

Page Miss       Before :         -1 -1 -1       After :  3 -1 -1
Page Miss       Before :         3 -1 -1        After :  3 4 -1
Page Hit
Page Miss       Before :         3 4 -1 After :  3 4 2

Hit ratio = 0.250000

1.FIFO
2.LEASTRECENTLYUSED
3.Optimal
4.Exit
Enter Choice : 2

Enter length of refererence string : 4

Enter refererence String with giving space
3 4 3 2

Page Miss       Before : -1 -1 -1       After : 3 -1 -1
Page Miss       Before : 3 -1 -1        After : 3 4 -1
Page Hit
Page Miss       Before : 3 4 -1 After : 3 4 2

Hit ratio = 0.250000
```



```
1.FIFO
2.LEASTRECENTLYUSED
3.Optimal
4.Exit
Enter Choice : 3

Enter length of refererence string : 4

Enter refererence String with giving space
3 4 3 2

Page Miss       Status : 3 -1 -1
Page Miss       Status : 3 4 -1
Page Hit
Page Miss       Status : 3 4 2

Hit ratio = 0.250000

1.FIFO
2.LEASTRECENTLYUSED
3.Optimal
4.Exit
Enter Choice : 4
```