# **Lab Assignment 7**: Page replacement

Name:Shivam Ganesh Gavandi
Roll no: 80
Class: TY-A

## 1)FIFO

```
#include<bits/stdc++.h>
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;
}
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 << pageFaults(pages, n, capacity);
    return 0;
}
```

**Output**
7

## 2)optimal page

```
int main()

{

   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10],
flag1, flag2, flag3, i, j, k, pos, max, faults = 0;

   printf("Enter number of frames: ");

   scanf("%d", &no_of_frames);


   printf("Enter number of pages: ");

   scanf("%d", &no_of_pages);


   printf("Enter page 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])
        {
            flag1 = flag2 = 1;
            break;
        }
    }

    if (flag1 == 0)
    {
        for (j = 0; j < no_of_frames; ++j)
        {
            if (frames[j] == -1)
            {
                faults++;
```

```
                frames[j] = pages[i];
                flag2 = 1;
                break;
            }
        }
    }

    if (flag2 == 0)
    {
        flag3 = 0;

        for (j = 0; j < no_of_frames; ++j)
        {
            temp[j] = -1;

            for (k = i + 1; k < no_of_pages; ++k)
            {
                if (frames[j] == pages[k])
                {
                    temp[j] = k;
                    break;
                }
            }
        }

        for (j = 0; j < no_of_frames; ++j)
        {
            if (temp[j] == -1)
            {
```

```
                pos = j;
                flag3 = 1;
                break;
            }
        }

        if (flag3 == 0)
        {
            max = temp[0];
            pos = 0;

            for (j = 1; j < no_of_frames; ++j)
            {
                if (temp[j] > max)
                {
                    max = temp[j];
                    pos = j;
                }
            }
        }
        frames[pos] = pages[i];
        faults++;
    }

    printf("\n");

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

```
        }

    }


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


    return 0;

}
```

**Output**

Enter number of frames: 3
Enter number of pages: 10
Enter page reference string: 2 3 4 2 1 3 7 5 4 3

2 -1 -1
2 3 -1
2 3 4
2 3 4
1 3 4
1 3 4
7 3 4
5 3 4
5 3 4

# 3)least recently used

```
#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;
                    }
                }

                s.erase(val);

                s.insert(pages[i]);

                page_faults++;
            }

            indexes[pages[i]] = i;
        }
    }

    return page_faults;
}

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 << pageFaults(pages, n, capacity);
    return 0;
}
```

Output=

6