LP 03

FIFO (First-In-First-Out)

```
#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[] = {1, 3, 0, 3, 5, 6, 3};
	int n = sizeof(pages)/sizeof(pages[0]);
	int capacity = 3;
	cout << pageFaults(pages, n, capacity);
	return 0;
}
```

OUTPUT :-

6

LP 03

LRU (Last Recently Used)

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

int main()
{
int capacity = 4;
int arr[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

deque<int> q(capacity);
int count=0;
int page_faults=0;
deque<int>::iterator itr;
q.clear();
for(int i:arr)
{
        itr = find(q.begin(),q.end(),i);
        if(!(itr != q.end()))
        {
        ++page_faults;
        if(q.size() == capacity)
        {
                q.erase(q.begin());
                q.push_back(i);
        }
        else{
                q.push_back(i);
        }
        }
        else
        {
        q.erase(itr);
        q.push_back(i);
        }
}
cout<<page_faults;
}
```

OUTPUT :-

6

LP 03

Optimal page replacement

```
#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]);
        int fn = 4;
        optimalPage(pg, pn, fn);
        return 0;
}
```

OUTPUT :-

```
No. of hits = 7
No. of misses = 6
```