Practical 7

1) Write a java program that implements the FIFO page-replacement algorithm.

1. **Package and Imports**

```
package com.mycompany.fifo2page;

import java.util.Arrays;
```

package com.mycompany.fifo2page;: This declares the package for the class FIFO2PAGE.
import java.util.Arrays;: This import is not used in the code but could be useful for array manipulation.

2. **Class and Method Definitions**

```
public class FIFO2PAGE {
```

public class FIFO2PAGE: Declares the public class named FIFO2PAGE.

checkHit Method

```
static boolean checkHit(int incomingPage, int[] queue, int occupied) {
    for (int i = 0; i < occupied; i++) {
        if (incomingPage == queue[i])
            return true;
    }
    return false;
}
```

checkHit: This method checks if a page is already in the queue.

- **incomingPage**: The page to check.
- **queue**: The array representing the current state of frames.
- **occupied**: The number of frames currently occupied.
- It returns true if the page is found in the queue, otherwise false.

printFrame Method

```
static void printFrame(int[] queue, int occupied) {
    for (int i = 0; i < occupied; i++)
        System.out.print(queue[i] + "\t\t\t");
}
```

printFrame: This method prints the current state of the frame queue.

- **queue**: The array of frames.
- **occupied**: The number of frames currently occupied.

3. **Main Method**

```
public static void main(String[] args) {
    int[] incomingStream = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3};
    int n = incomingStream.length;
    int frames = 3;
    int[] queue = new int[frames];
    int[] distance = new int[frames];
    int occupied = 0;
    int pagefault = 0;

    System.out.println("Page\t Frame1 \t Frame2 \t Frame3");
```

incomingStream: Represents the sequence of page references.
n: The length of the page reference stream.
frames: The number of frames available in memory.
queue: An array representing the current state of the page frames.
distance: An array used to store the distance of pages from the current position.
occupied: Keeps track of the number of frames currently occupied.
pagefault: Counts the number of page faults that occur.

```
for (int i = 0; i < n; i++) {
    System.out.print(incomingStream[i] + ": \t\t");

    if (checkHit(incomingStream[i], queue, occupied)) {
        printFrame(queue, occupied);
    } else if (occupied < frames) {
        queue[occupied] = incomingStream[i];
        pagefault++;
        occupied++;
        printFrame(queue, occupied);
    } else {
        int max = Integer.MIN_VALUE;
        int index = -1;
        for (int j = 0; j < frames; j++) {
            distance[j] = 0;
            for (int k = i - 1; k >= 0; k--) {
                ++distance[j];
                if (queue[j] == incomingStream[k])
```

```
                break;
            }
            if (distance[j] > max) {
                max = distance[j];
                index = j;
            }
        }
        queue[index] = incomingStream[i];
        printFrame(queue, occupied);
        pagefault++;
    }
    System.out.println();
  }
  System.out.println("Page Fault: " + pagefault);
}
```

for (int i = 0; i < n; i++): Iterates through each page in the incoming stream.

- **if (checkHit(incomingStream[i], queue, occupied))**: Checks if the current page is already in the queue (no page fault).
- **else if (occupied < frames)**: If there are empty frames, simply add the page to the queue.
- **else**: If the frames are full, use the optimal page replacement strategy:
    - **Determine which page to replace**: Calculate the distance each page in the queue will be used in the future. Replace the page that is used furthest in the future.
- **printFrame(queue, occupied)**: Print the state of the frames after each operation.

- **pagefault++**: Increment the page fault count each time a page is added or replaced.

Conclusion:

This code simulates the Optimal Page Replacement Algorithm, not FIFO, with a fixed number of page frames. It keeps track of page hits and page faults as it processes a sequence of page references. The key points are:

- **Page Hit**: When the page is already in one of the frames.
- **Page Fault**: When the page is not in the frames, requiring either insertion or replacement.
- **Optimal Replacement**: Chooses the page that will be used farthest in the future for replacement when needed.

The algorithm prints the state of the page frames after each page reference and finally outputs the total number of page faults.

Code

```
package com.mycompany.fifo2page;


import java.util.Arrays;


public class FIFO2PAGE {
    static boolean checkHit(int incomingPage, int[] queue, int occupied)
    {
```

```
    for (int i = 0; i < occupied; i++)
    {
        if (incomingPage == queue[i])
            return true;
    }
    return false;
}

static void printFrame(int[] queue, int occupied)
{
    for (int i = 0; i < occupied; i++)
        System.out.print(queue[i] + "\t\t\t");
}

public static void main(String[] args) {
            int[] incomingStream = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3};
    int n = incomingStream.length;
    int frames = 3;
    int[] queue = new int[frames];
    int[] distance = new int[frames];
    int occupied = 0;
    int pagefault = 0;

    System.out.println("Page\t Frame1 \t Frame2 \t Frame3");
```

```
for (int i = 0; i < n; i++) {
    System.out.print(incomingStream[i] + ": \t\t");

    if (checkHit(incomingStream[i], queue, occupied))
    {
        printFrame(queue, occupied);
    } else if (occupied < frames)
    {
        queue[occupied] = incomingStream[i];
        pagefault++;
        occupied++;
        printFrame(queue, occupied);
    } else
    {
        int max = Integer.MIN_VALUE;
        int index = -1;
        for (int j = 0; j < frames; j++)
        {
            distance[j] = 0;
            for (int k = i - 1; k >= 0; k--)
            {
                ++distance[j];
                if (queue[j] == incomingStream[k])
                    break;
            }
```

```
                if (distance[j] > max)
                {
                    max = distance[j];
                    index = j;
                }
            }
            queue[index] = incomingStream[i];
            printFrame(queue, occupied);
            pagefault++;
        }
        System.out.println();
    }
    System.out.println("Page Fault: " + pagefault);
  }
}
```