import java.util.\*;

public class PageReplacement {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (true) {

// Accept frame size and reference string from user

System.out.print("Enter the number of frames: ");

int frameSize = scanner.nextInt();

System.out.print("Enter the number of pages: ");

int n = scanner.nextInt();

int[] referenceString = new int[n];

System.out.print("Enter the reference string (space-separated): ");

for (int i = 0; i < n; i++) {

referenceString[i] = scanner.nextInt();

}

// Select algorithm

System.out.println("\nSelect the page replacement algorithm(s):");

System.out.println("1. FIFO");

System.out.println("2. Optimal");

System.out.println("3. Both FIFO and Optimal");

System.out.println("4. Exit");

System.out.print("Enter choice (1/2/3/4): ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

fifoPageReplacement(referenceString, frameSize);

break;

case 2:

optimalPageReplacement(referenceString, frameSize);

break;

case 3:

// Run both FIFO and Optimal algorithms

fifoPageReplacement(referenceString, frameSize);

optimalPageReplacement(referenceString, frameSize);

break;

case 4:

System.out.println("Exiting the program...");

scanner.close();

return; // Exit the program

default:

System.out.println("Invalid choice. Please select again.");

}

}

}

// FIFO Page Replacement Algorithm

private static void fifoPageReplacement(int[] referenceString, int frameSize) {

Set<Integer> frames = new LinkedHashSet<>();

Queue<Integer> fifoQueue = new LinkedList<>();

int pageFaults = 0;

for (int page : referenceString) {

if (!frames.contains(page)) {

if (frames.size() == frameSize) {

int oldestPage = fifoQueue.poll();

frames.remove(oldestPage);

}

frames.add(page);

fifoQueue.add(page);

pageFaults++;

}

}

System.out.println("Total page faults (FIFO): " + pageFaults);

}

// Optimal Page Replacement Algorithm

private static void optimalPageReplacement(int[] referenceString, int frameSize) {

Set<Integer> frames = new LinkedHashSet<>();

int pageFaults = 0;

for (int i = 0; i < referenceString.length; i++) {

int page = referenceString[i];

if (!frames.contains(page)) {

if (frames.size() == frameSize) {

int pageToReplace = getOptimalPageToReplace(referenceString, frames, i);

frames.remove(pageToReplace);

}

frames.add(page);

pageFaults++;

}

}

System.out.println("Total page faults (Optimal): " + pageFaults);

}

// Method to find the page to replace for Optimal algorithm

private static int getOptimalPageToReplace(int[] referenceString, Set<Integer> frames, int currentIndex) {

int farthestUse = -1;

int pageToReplace = -1;

// Iterate through each page in frames and determine which is used the farthest in the future

for (int page : frames) {

int nextUse = getNextUse(referenceString, currentIndex, page);

if (nextUse == -1) {

// If a page is not used in the future, replace it

return page;

}

if (nextUse > farthestUse) {

farthestUse = nextUse;

pageToReplace = page;

}

}

return pageToReplace;

}

// Method to find the next use index of a page in the reference string

private static int getNextUse(int[] referenceString, int currentIndex, int page) {

for (int i = currentIndex + 1; i < referenceString.length; i++) {

if (referenceString[i] == page) {

return i;

}

}

return -1; // Return -1 if the page is not used again

}

}