import java.util.Scanner;

class LinearProbingHashTable

{

private int currentSize, maxSize;

private String[] keys;

private String[] vals;

public LinearProbingHashTable(int capacity)

{

currentSize = 0;

maxSize = capacity;

keys = new String[maxSize];

vals = new String[maxSize];

}

public void makeEmpty()

{

currentSize = 0;

keys = new String[maxSize];

vals = new String[maxSize];

}

public int getSize()

{

return currentSize;

}

public boolean isFull()

{

return currentSize == maxSize;

}

public boolean isEmpty()

{

return getSize() == 0;

}

public boolean contains(String key)

{

return get(key) != null;

}

private int hash(String key)

{

return key.hashCode() % maxSize;

}

public void insert(String key, String val)

{

int tmp = hash(key);

int i = tmp;

do

{

if (keys[i] == null)

{

keys[i] = key;

vals[i] = val;

currentSize++;

return;

}

if (keys[i].equals(key))

{

vals[i] = val;

return;

}

i = (i + 1) % maxSize;

} while (i != tmp);

}

public String get(String key)

{

int i = hash(key);

while (keys[i] != null)

{

if (keys[i].equals(key))

return vals[i];

i = (i + 1) % maxSize;

}

return null;

}

public void remove(String key)

{

if (!contains(key))

return;

int i = hash(key);

while (!key.equals(keys[i]))

i = (i + 1) % maxSize;

keys[i] = vals[i] = null;

for (i = (i + 1) % maxSize; keys[i] != null; i = (i + 1) % maxSize)

{

String tmp1 = keys[i], tmp2 = vals[i];

keys[i] = vals[i] = null;

currentSize--;

insert(tmp1, tmp2);

}

currentSize--;

}

public void printHashTable()

{

System.out.println("\nHash Table: ");

for (int i = 0; i < maxSize; i++)

if (keys[i] != null)

System.out.println(keys[i] +" "+ vals[i]);

System.out.println();

}

}

public class LinearProbingHashTableTest

{

public static void main(String[] args)

{

Scanner scan = new Scanner(System.in);

System.out.println("Hash Table Test\n\n");

System.out.println("Enter size");

LinearProbingHashTable lpht = new LinearProbingHashTable(scan.nextInt() );

char ch;

do

{

System.out.println("\nHash Table Operations\n");

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

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

System.out.println("3. get");

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

System.out.println("5. size");

int choice = scan.nextInt();

switch (choice)

{

case 1 :

System.out.println("Enter key and value");

lpht.insert(scan.next(), scan.next() );

break;

case 2 :

System.out.println("Enter key");

lpht.remove( scan.next() );

break;

case 3 :

System.out.println("Enter key");

System.out.println("Value = "+ lpht.get( scan.next() ));

break;

case 4 :

lpht.makeEmpty();

System.out.println("Hash Table Cleared\n");

break;

case 5 :

System.out.println("Size = "+ lpht.getSize() );

break;

default :

System.out.println("Wrong Entry \n ");

break;

}

lpht.printHashTable();

System.out.println("\nDo you want to continue (Type y or n) \n");

ch = scan.next().charAt(0);

} while (ch == 'Y'|| ch == 'y');

}

}

import java.util.Arrays;

import java.util.Random;

public class SortTimer {

   // Please expand method main() to meet the lab requirements.

   // You have the following sorting methods available:

   // insertionSort(int[] a);

   // selectionSort(int[] a);

   // mergeSort(int[] a);

   // quickSort(int[] a);

   // The array will be in sorted order after the routines are called!

   // Be sure to re-randomize the array after each sort.

    public static void InsertNumbersToArray(int[] a){

        Random rand = new Random(); //instance of random class

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

            a[i] = rand.nextInt(10000);

        }

    }

    public static long MeasureTime(int[] a, int sortType){

        long startTime = System.currentTimeMillis();

        switch(sortType){

            case 1: quickSort(a);

                    break;

            case 2: mergeSort(a);

                    break;

            case 3: insertionSort(a);

                    break;

            case 4: selectionSort(a);

        }

        long endTime = System.currentTimeMillis();

        long duration = (endTime - startTime);

        return duration;

    }

    public static void randomizeArray( int arr[])

    {

        // Creating a object for Random class

        Random r = new Random();

        // Start from the last element and swap one by one. We don't

        // need to run for the first element that's why i > 0

        for (int i = arr.length-1; i > 0; i--) {

            // Pick a random index from 0 to i

            int j = r.nextInt(i+1);

            // Swap arr[i] with the element at random index

            int temp = arr[i];

            arr[i] = arr[j];

            arr[j] = temp;

        }

    }

    public static void Testing(int[] a){

        long duration;

        System.out.println("\nTesting an array of length " + a.length + ".");

        duration = MeasureTime(a, 1);

        System.out.println("Quick sort:" + duration + "ms");

        randomizeArray(a);

        duration = MeasureTime(a, 2);

        System.out.println("Merge sort:" + duration + "ms");

        randomizeArray(a);

        duration = MeasureTime(a, 3);

        System.out.println("Insertion sort:" + duration + "ms");

        randomizeArray(a);

        duration = MeasureTime(a, 4);

        System.out.println("Selection sort:" + duration + "ms");

        randomizeArray(a);

    }

   public static void main(String[] args) {

       // Creating 5 arrays

       int[] a = new int[100];

       int[] b = new int[1000];

       int[] c = new int[10000];

       int[] d = new int[100000];

       int[] e = new int[1000000];

       InsertNumbersToArray(a);

       InsertNumbersToArray(b);

       InsertNumbersToArray(c);

       InsertNumbersToArray(d);

       InsertNumbersToArray(e);

       Testing(a);

       Testing(b);

       Testing(c);

       Testing(d);

       Testing(e);

   }

}