A paper with text and numbers

Description automatically generated

A paper with text and words

Description automatically generated

Counting Sort code:

import java.util.Vector;

public class CountingSort{

    public static void counting\_sort(Vector<Integer> data, Vector<Integer> sortedData, Integer max){

        Vector<Integer> count = new Vector<Integer>();

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

            count.add(0);

        }

        for (int i = 0; i < data.size(); i++){

            count.set(data.elementAt(i), count.elementAt(data.elementAt(i)) + 1);

        }

        for (int i = 1; i <= max; i++){

            count.set(i, count.elementAt(i) + count.elementAt(i - 1));

        }

        for (int i = data.size() - 1; i >= 0; i--){

            sortedData.add(0);

        }

        for (int i = data.size() - 1; i >= 0; i--){

            sortedData.set(count.elementAt(data.elementAt(i)) - 1, data.elementAt(i));

            count.set(data.elementAt(i), count.elementAt(data.elementAt(i)) - 1);

        }

    }

    public static void main (String[] args){

        Vector<Integer> myData = new Vector<Integer>();

        myData.add(11);

        myData.add(7);

        myData.add(12);

        myData.add(18);

        myData.add(21);

        myData.add(15);

        myData.add(14);

        myData.add(4);

        System.out.println("Original Data: " + myData);

        Integer max = myData.elementAt(0);

        for (int i = 1; i < myData.size(); i++){

            if (myData.elementAt(i) > max){

                max = myData.elementAt(i);

            }

        }

        Vector <Integer> mySortedData = new Vector<Integer>();

        counting\_sort(myData, mySortedData, max);

        System.out.println("Sorted Data: " + mySortedData);

    }

}

CountingSort output:

A screenshot of a computer program

Description automatically generated

CountingSortOptimization code:

// Implement the given algorithm in Java, and now generalize it to work for generic data. Your implementation

// needs to be able to sort collections of user-defined objects, given that the following operators have been defined

// for the object:

// • maximal index corresponding to maximal value: k in the integer case as input to the given algorithm. For

// example, if you were using the capital letters of the English alphabet, you may choose Z, being 25 (if A is

// 0) to represent the maximal value, and 25 to be the index corresponding to it.

// • integer index corresponding to the value of the user-defined object. This is similar to maximal index, but

// this is the general case. This is used in lines 5, 11, and 12 of the given algorithm.

// • default value: 0 in the integer case in line 3 of the given algorithm. Use a default constructor here.

// • assignment =

// • increment ++

// • decrement --

// • addition +

import java.util.\*;

public class CountingSortOptimization {

    public static <T extends Comparable<? super T>> void counting\_sort(Vector<T> data, Vector<T> sortedData, T max) {

        HashMap<T, Integer> count = new HashMap<>();

        // Initialize count map

        for (T element : data) {

            count.put(element, 0);

        }

        // Count occurrences of each element

        for (T element : data) {

            count.put(element, count.get(element) + 1);

        }

        // Calculate cumulative counts

        ArrayList<T> keys = new ArrayList<>(count.keySet());

        Collections.sort(keys);

        for (int i = 1; i < keys.size(); i++) {

            count.put(keys.get(i), count.get(keys.get(i)) + count.get(keys.get(i - 1)));

        }

        // Build the sorted output array

        for (int i = data.size() - 1; i >= 0; i--) {

            sortedData.add(null); // Adjust size to avoid IndexOutOfBoundsException

        }

        for (int i = data.size() - 1; i >= 0; i--) {

            sortedData.set(count.get(data.get(i)) - 1, data.get(i));

            count.put(data.get(i), count.get(data.get(i)) - 1);

        }

    }

    public static void main(String[] args) {

        Vector<String> myData = new Vector<>();

        myData.add("banana");

        myData.add("apple");

        myData.add("orange");

        myData.add("grape");

        myData.add("kiwi");

        myData.add("pear");

        myData.add("strawberry");

        myData.add("blueberry");

        System.out.println("Original Data: " + myData);

        String max = Collections.max(myData);

        Vector<String> mySortedData = new Vector<>();

        counting\_sort(myData, mySortedData, max);

        System.out.println("Sorted Data: " + mySortedData);

    }

}

CountingSortOptimization output:

A screen shot of a computer program

Description automatically generated

All code works as intended!