class SelectionSort {

void selectionSort(int array[]) {

int size = array.length;

for (int step = 0; step < size - 1; step++) {

int min\_idx = step;

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

// To sort in descending order, change > to < in this line.

// Select the minimum element in each loop.

if (array[i] < array[min\_idx]) {

min\_idx = i;

}

}

// put min at the correct position

int temp = array[step];

array[step] = array[min\_idx];

array[min\_idx] = temp;

}

}

// Bubble sort in Java

import java.util.Arrays;

class Main {

// perform the bubble sort

static void bubbleSort(int array[]) {

int size = array.length;

// loop to access each array element

for (int i = 0; i < size - 1; i++)

// loop to compare array elements

for (int j = 0; j < size - i - 1; j++)

// compare two adjacent elements

// change > to < to sort in descending order

if (array[j] > array[j + 1]) {

// swapping occurs if elements

// are not in the intended order

int temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

// Insertion sort in Java

import java.util.Arrays;

class InsertionSort {

void insertionSort(int array[]) {

int size = array.length;

for (int step = 1; step < size; step++) {

int key = array[step];

int j = step - 1;

// Compare key with each element on the left of it until an element smaller than

// it is found.

// For descending order, change key<array[j] to key>array[j].

while (j >= 0 && key < array[j]) {

array[j + 1] = array[j];

--j;

}

// Place key at after the element just smaller than it.

array[j + 1] = key;

}

}

public class Main {

public static void countSort(int[] arr, int min, int max) {

int range = max - min + 1;

int[] ans = new int[arr.length];

//make frequency arr

int[] farr = new int[range];

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

farr[arr[i] - min]++;

}

//convert it into prefix sum array

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

farr[i] += farr[i - 1];

}

//stable sorting(filling ans array)

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

int pos = farr[arr[i] - min] - 1;

ans[pos] = arr[i];

farr[arr[i] - min]--;

}

//filling original array with the help of ans array

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

arr[i] = ans[i];

}

}

public static void print(int[] arr) {

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

System.out.println(arr[i]);

}

}

import java.io.\*;

import java.util.\*;

//[n-1] 3 5 1 2

/\*

3 5 1 2

3 5 1 2

3 1 5 2

for(int i=0;i<a.length-1;i++)

{

for(int j=i;j<a.length-i-1;j++)

{

}

}

\*/

public class Bubble {

public static void main(String[] args) throws Exception

{

int a[]={4, 2, 2, 8, 3, 3, 1};

int n = a.length;

System.out.println("=="+a.length);

// insert(a);

// selection(a);

//select(a);

count(a,1,8);

}

public static void count(int a[],int min,int max)

{

int range=max-min+1;

int freq[]=new int[range];

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

{

int idx=a[i]-min;

freq[idx]++;

}

display(freq);

System.out.println("===");

for(int i=1;i<freq.length;i++)

{

freq[i]+=freq[i-1];

//freq[i]++;

}

display(freq);

int ans[]=new int[a.length];

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

{

int pos=freq[a[i]-min]-1;

ans[pos]=a[i];

freq[a[i]-min]--;

}

System.out.println("===========");

display(ans);

}

public static void select(int a[])

{

for(int i=0;i<a.length-1;i++)

{

int min=i;

for(int j=i+1;j<a.length;j++)

{

if(a[min]>a[j])

{

min=j;

}

}

int t=a[min];

a[min]=a[i];

a[i]=t;

}

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

{

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

}

}

public static void bb(int[] a)

{

int n=a.length;

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-i-1;j++)

{

if(a[j]>a[j+1])

{

int temp=a[j+1];

a[j+1]=a[j];

a[j]=temp;

}

}

}

}

public static void display(int a[])

{

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

{

System.out.println(i+"="+a[i]);

}

}

public static void insert(int a[])

{

int lgt=a.length;

for(int i=1;i<lgt;i++)

{

for(int j=i-1; j>=0;j--)

{

if(a[j]>a[j+1])

{

int t=a[j];

a[j]=a[j+1];

a[j+1]=t;

}

}

}

display(a);

}

}

class BinarySearch

{

    // Returns index of x if it is present in arr[l..

    // r], else return -1

    int binarySearch(int arr[], int l, int r, int x)

    {

        if (r>=l)

        {

            int mid = l + (r - l)/2;

            // If the element is present at the

            // middle itself

            if (arr[mid] == x)

               return mid;

            // If element is smaller than mid, then

            // it can only be present in left subarray

            if (arr[mid] > x)

               return binarySearch(arr, l, mid-1, x);

            // Else the element can only be present

            // in right subarray

            return binarySearch(arr, mid+1, r, x);

        }

        // We reach here when element is not present

        //  in array

        return -1;

    }