**Министерство образования и науки Российской Федерации**

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,   
МЕХАНИКИ И ОПТИКИ**

Факультет программной инженерии и компьютерной техники

Кафедра информатики и прикладной математики   
Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Алгоритмы и структуры данных»

**ОТЧЁТ**

по лабораторной работе №2  
неделя вторая

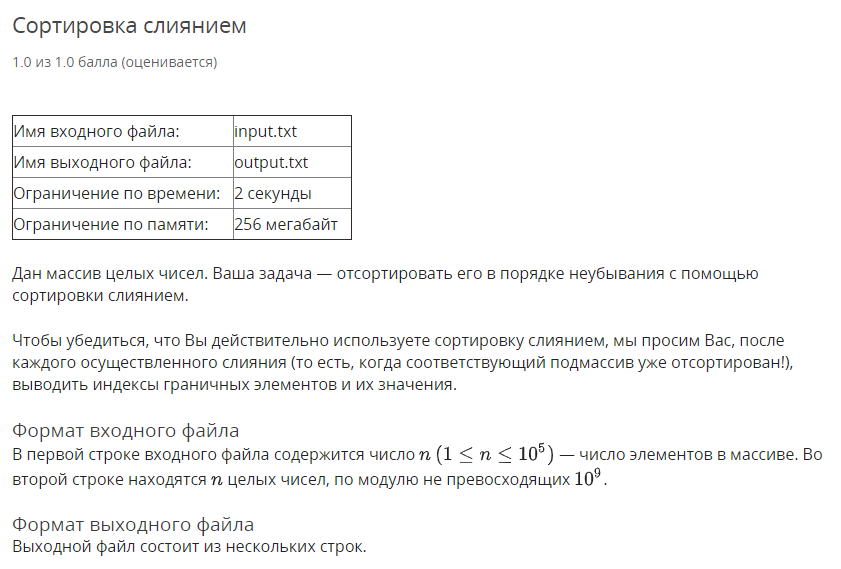
Выполнил:  
Айгузин Иван Олегович   
P3218

Преподаватели:

Романов Алексей Андреевич  
Волчек Дмитрий Геннадьевич

Санкт-Петербург

2018



using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace Task01 {

public sealed class Program {

private static StreamReader \_in;

private static StreamWriter \_out;

private static void Main(string[] args) {

if (!args.Contains("console")) {

SetupIO();

}

Run();

if (args.Contains("console")) {

Console.ReadLine();

}

DisposeIO();

}

private static void Run() {

var n = int.Parse(Console.ReadLine());

var numbers = Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToList();

var sortInfo = new MergeSorter().Sort(numbers);

// Result

sortInfo.Splits

.Select(x => $"{x.leftIndex + 1} {x.rightIndex + 1} {x.leftValue} {x.rightValue}")

.ToList()

.ForEach(Console.WriteLine);

Console.WriteLine(string.Join(" ", sortInfo.ResultingArray));

}

private static void SetupIO() {

\_in = new StreamReader("input.txt");

\_out = new StreamWriter("output.txt");

Console.SetIn(\_in);

Console.SetOut(\_out);

}

private static void DisposeIO() {

\_in?.Dispose();

\_out?.Dispose();

}

}

internal class MergeSorter {

public SortInfo<T> Sort<T>(IEnumerable<T> input) where T : IComparable<T> {

var sortInfo = new SortInfo<T>(input.ToArray());

var splits = sortInfo.Splits;

var array = sortInfo.ResultingArray;

SortPart(0, array.Length - 1);

return sortInfo;

void SortPart(int left, int right) {

if (right <= left) {

return;

}

var mid = (left + right) / 2;

SortPart(left, mid);

SortPart(mid + 1, right);

MergePart(left, mid, right);

splits.Add((left, right, array[left], array[right]));

}

void MergePart(int left, int mid, int right) {

var buffer = new List<T>(right - left + 1);

var leftCursor = left;

var rightCursor = mid + 1;

for (var i = 0; i < buffer.Capacity; i++) {

if (leftCursor <= mid && rightCursor <= right) {

if (array[leftCursor].CompareTo(array[rightCursor]) <= 0) {

buffer.Add(array[leftCursor]);

leftCursor += 1;

}

else {

buffer.Add(array[rightCursor]);

rightCursor += 1;

}

continue;

}

if (leftCursor <= mid) {

buffer.Add(array[leftCursor]);

leftCursor += 1;

continue;

}

buffer.Add(array[rightCursor]);

rightCursor += 1;

}

for (var i = 0; i < buffer.Count; i++) {

array[left + i] = buffer[i];

}

}

}

internal class SortInfo<T> {

public SortInfo(T[] resultingArray) {

ResultingArray = resultingArray;

}

public List<(int leftIndex, int rightIndex, T leftValue, T rightValue)> Splits { get; }

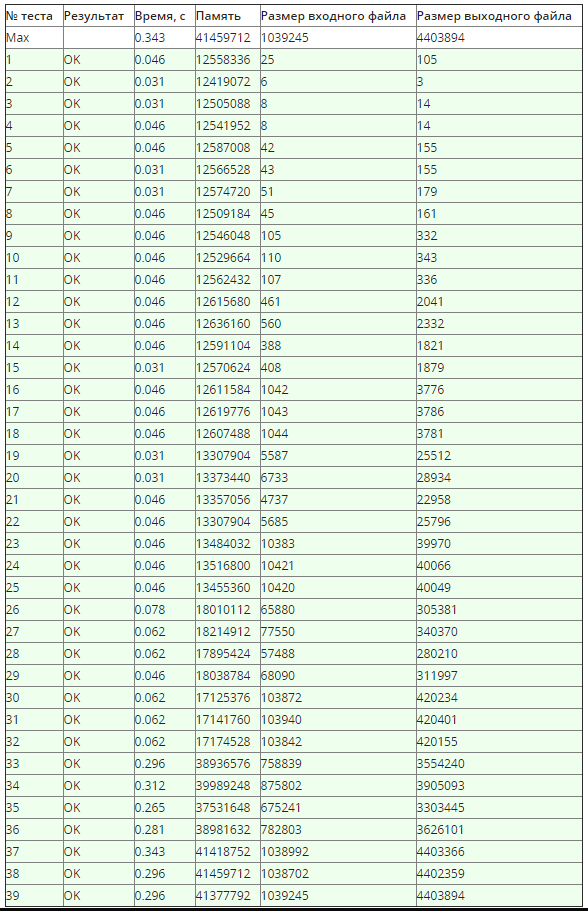
= new List<(int leftIndex, int rightIndex, T leftValue, T rightValue)>();

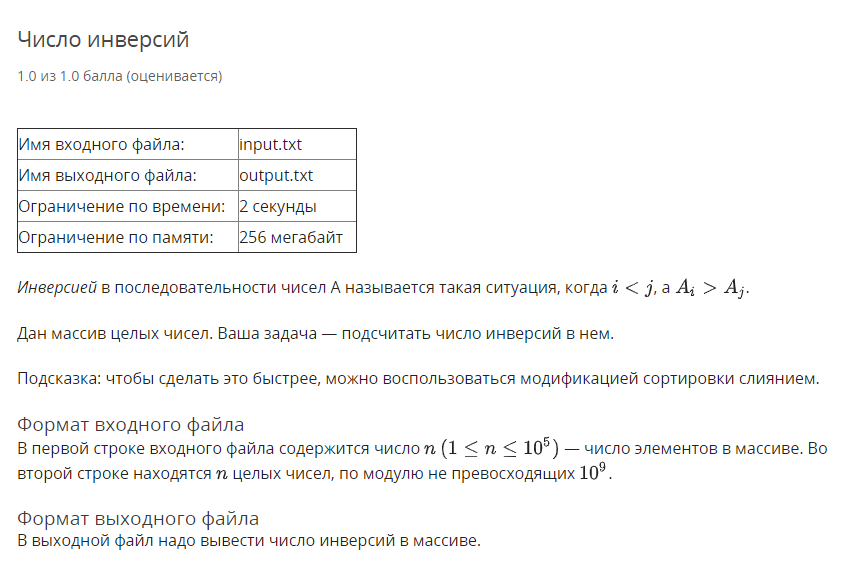
public T[] ResultingArray { get; }

}

}

}





using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace Task02 {

public sealed class Program {

private static StreamReader \_in;

private static StreamWriter \_out;

private static void Main(string[] args) {

if (!args.Contains("console")) {

SetupIO();

}

Run();

if (args.Contains("console")) {

Console.ReadLine();

}

DisposeIO();

}

private static void Run() {

var n = int.Parse(Console.ReadLine());

var numbers = Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToList();

var sortInfo = new MergeSorter().Sort(numbers);

Console.WriteLine(sortInfo.InverseAmount);

}

private static void SetupIO() {

\_in = new StreamReader("input.txt");

\_out = new StreamWriter("output.txt");

Console.SetIn(\_in);

Console.SetOut(\_out);

}

private static void DisposeIO() {

\_in?.Dispose();

\_out?.Dispose();

}

}

internal class MergeSorter {

public SortInfo Sort<T>(IEnumerable<T> input) where T : IComparable<T> {

var inverseAmount = 0L;

var array = input.ToArray();

SortPart(0, array.Length - 1);

return new SortInfo(inverseAmount);

void SortPart(int left, int right) {

if (right <= left) {

return;

}

var mid = (left + right) / 2;

SortPart(left, mid);

SortPart(mid + 1, right);

MergePart(left, mid, right);

}

void MergePart(int left, int mid, int right) {

var buffer = new List<T>(right - left + 1);

var leftCursor = left;

var rightCursor = mid + 1;

for (var i = 0; i < buffer.Capacity; i++) {

if (leftCursor <= mid && rightCursor <= right) {

if (array[leftCursor].CompareTo(array[rightCursor]) <= 0) {

buffer.Add(array[leftCursor]);

leftCursor += 1;

}

else {

buffer.Add(array[rightCursor]);

rightCursor += 1;

inverseAmount += mid - leftCursor + 1; //

}

continue;

}

if (leftCursor <= mid) {

buffer.Add(array[leftCursor]);

leftCursor += 1;

continue;

}

buffer.Add(array[rightCursor]);

rightCursor += 1;

}

for (var i = 0; i < buffer.Count; i++) {

array[left + i] = buffer[i];

}

}

}

internal class SortInfo {

public SortInfo(long inverseAmount) {

InverseAmount = inverseAmount;

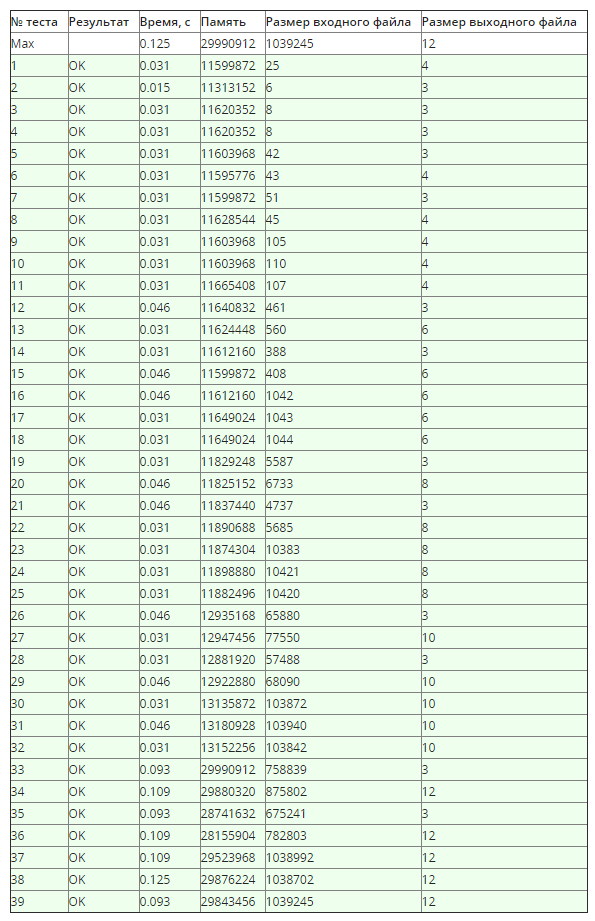
}

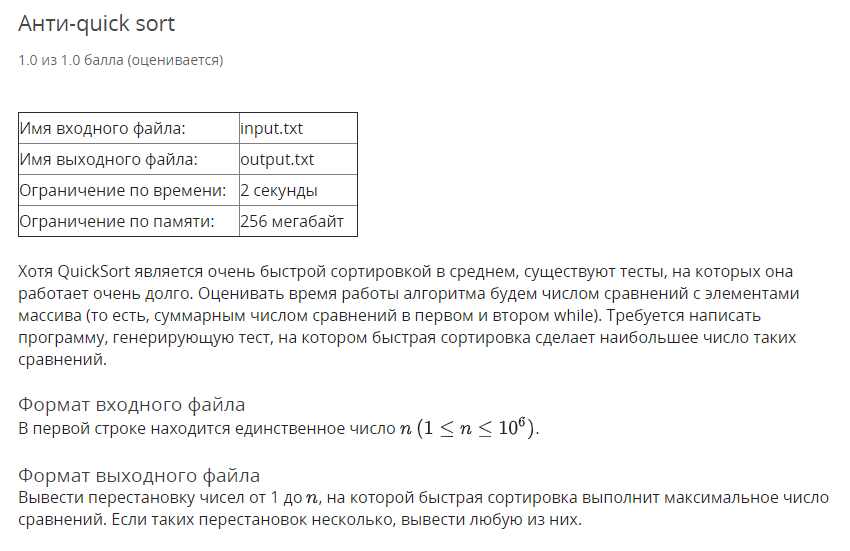
public long InverseAmount { get; }

}

}

}





using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace Task03 {

public sealed class Program {

private static StreamReader \_in;

private static StreamWriter \_out;

private static void Main(string[] args) {

if (!args.Contains("console")) {

SetupIO();

}

Run();

if (args.Contains("console")) {

Console.ReadLine();

}

DisposeIO();

}

private static void Run() {

var n = int.Parse(Console.ReadLine());

var numbers = new List<int>(n);

for (var i = 1; i <= n; i++) {

numbers.Add(i);

var mid = (i - 1) / 2;

if (i > 2) {

(numbers[i - 1], numbers[mid]) = (numbers[mid], numbers[i - 1]);

}

}

Console.WriteLine(string.Join(" ", numbers));

}

private static void SetupIO() {

\_in = new StreamReader("input.txt");

\_out = new StreamWriter("output.txt");

Console.SetIn(\_in);

Console.SetOut(\_out);

}

private static void DisposeIO() {

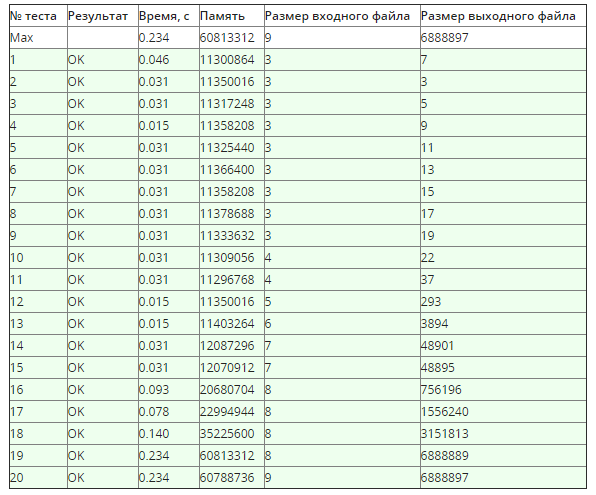
\_in?.Dispose();

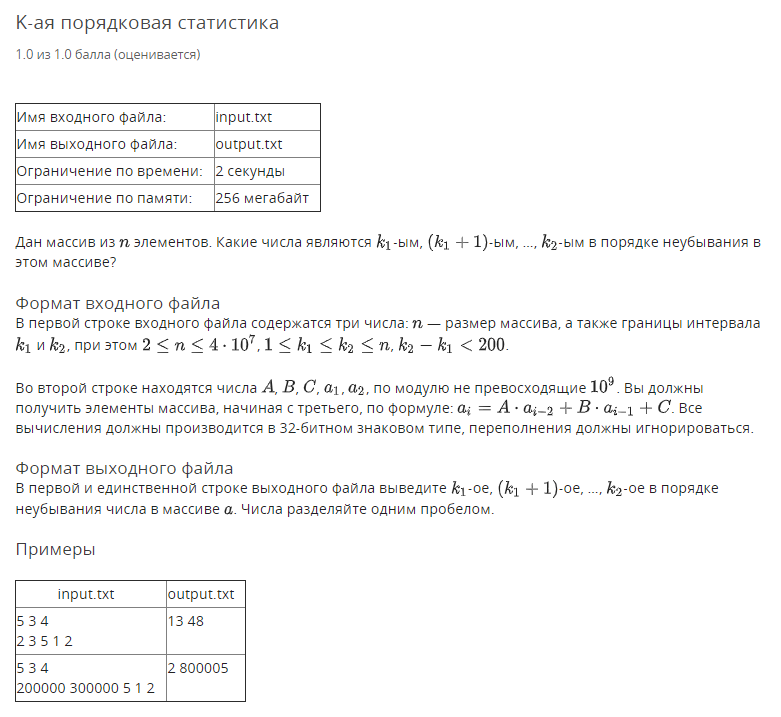
\_out?.Dispose();

}

}

}





using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace Task03 {

public sealed class Program {

private static StreamReader \_in;

private static StreamWriter \_out;

private static void Main(string[] args) {

if (!args.Contains("console")) {

SetupIO();

}

Run();

if (args.Contains("console")) {

Console.ReadLine();

}

DisposeIO();

}

private static void Run() {

var (n, k1, k2) = ReadFirstLine();

var numbers = GenerateNumbers(n);

QuickSort(numbers, 0, n - 1, k1, k2);

for (var i = k1 - 1; i < k2; i++) {

Console.Write($"{numbers[i]} ");

}

}

private static void QuickSort<T>(IList<T> elements, int left, int right, int k1, int k2)

where T : IComparable<T> {

while (true) {

if (left >= right || left > k2 - 1 || right < k1 - 1) {

return;

}

var i = left;

var j = right;

var pivot = elements[(left + right) / 2];

while (i <= j) {

while (elements[i].CompareTo(pivot) < 0) {

i++;

}

while (elements[j].CompareTo(pivot) > 0) {

j--;

}

if (i > j) {

continue;

}

(elements[i], elements[j]) = (elements[j], elements[i]);

i++;

j--;

}

QuickSort(elements, left, j, k1, k2);

left = i; // recursion -> iteration

}

}

private static List<int> GenerateNumbers(int n) {

var (A, B, C, a1, a2) = ReadSecondLine();

var numbers = new List<int>(n) {a1, a2};

for (var i = 2; i < n; i++) {

var a = unchecked(A \* numbers[i - 2] + B \* numbers[i - 1] + C);

numbers.Add(a);

}

return numbers;

}

private static (int n, int k1, int k2) ReadFirstLine() {

var p = ReadIntList();

return (p[0], p[1], p[2]);

}

private static (int A, int B, int C, int a1, int a2) ReadSecondLine() {

var p = ReadIntList();

return (p[0], p[1], p[2], p[3], p[4]);

}

private static List<int> ReadIntList() {

return Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToList();

}

private static void SetupIO() {

\_in = new StreamReader("input.txt");

\_out = new StreamWriter("output.txt");

Console.SetIn(\_in);

Console.SetOut(\_out);

}

private static void DisposeIO() {

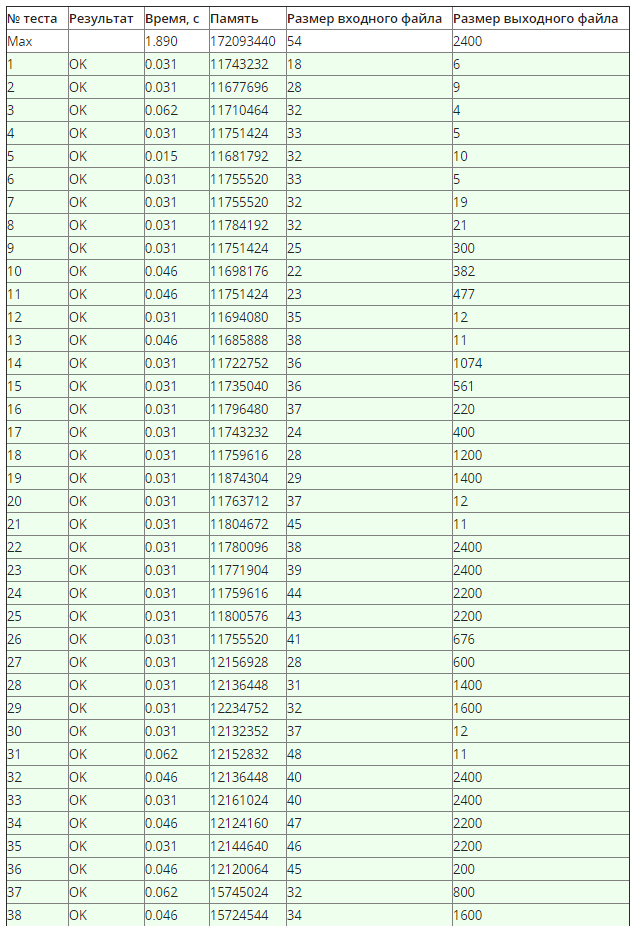
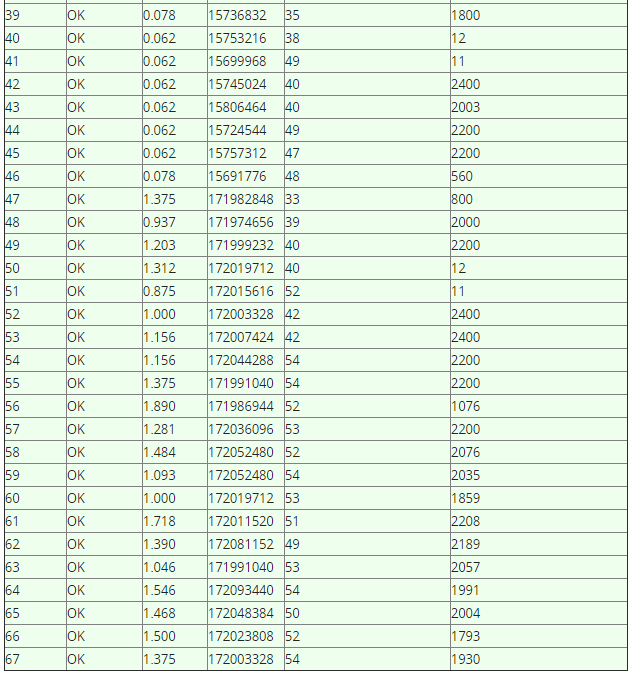
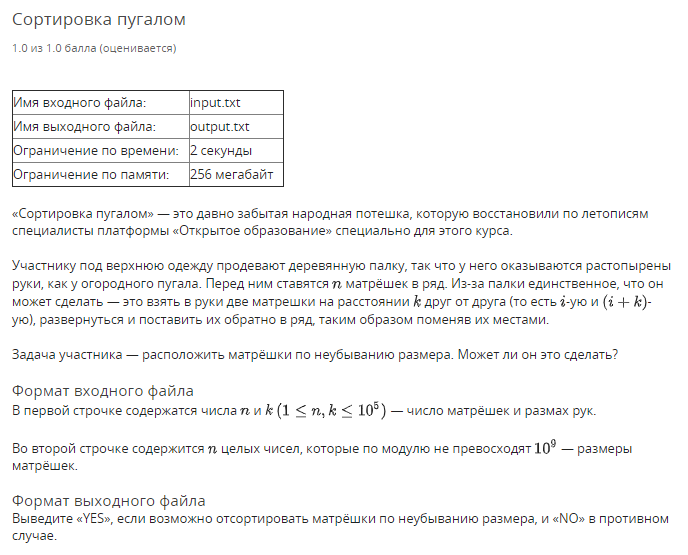
\_in?.Dispose();

\_out?.Dispose();

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace Task05 {

public sealed class Program {

private static StreamReader In;

private static StreamWriter Out;

private static void Main(string[] args) {

if (!args.Contains("console")) {

SetupIO();

}

Run();

if (args.Contains("console")) {

Console.ReadLine();

}

DisposeIO();

}

private static void Run() {

var (n, k) = ReadFirstLine();

var numbers = ReadIntList();

Sort(numbers, n, k);

var notSorted = Enumerable.Range(0, n - 1)

.Any(i => numbers[i] > numbers[i + 1]);

Console.WriteLine(notSorted ? "NO" : "YES");

}

private static void Sort(List<int> numbers, int n, int k) {

if (k == 1) {

QuickSort(numbers, 0, n - 1, k);

return;

}

var offset = n - n % k;

for (var i = 0; i < k; i++) {

var cursor = i + offset;

QuickSort(numbers, i, cursor < n ? cursor : cursor - k, k);

}

}

private static void QuickSort<T>(IList<T> elements, int left, int right, int k) where T : IComparable<T> {

while (true) {

if (left >= right) {

return;

}

var i = left;

var j = right;

var pivot = elements[(left + right) / (k \* 2) \* k + left % k];

while (i <= j) {

while (elements[i].CompareTo(pivot) < 0) {

i += k;

}

while (elements[j].CompareTo(pivot) > 0) {

j -= k;

}

if (i > j) {

continue;

}

(elements[i], elements[j]) = (elements[j], elements[i]);

i += k;

j -= k;

}

QuickSort(elements, left, j, k);

left = i; // recursion -> iteration

}

}

private static (int N, int K) ReadFirstLine() {

var p = ReadIntList();

return (p[0], p[1]);

}

private static List<int> ReadIntList() {

return Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToList();

}

private static void SetupIO() {

In = new StreamReader("input.txt");

Out = new StreamWriter("output.txt");

Console.SetIn(In);

Console.SetOut(Out);

}

private static void DisposeIO() {

In?.Dispose();

Out?.Dispose();

}

}

}

