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

Сортировка слиянием

1.0 из 1.0 балла (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубывания с помощью сортировки слиянием.

Чтобы убедиться, что Вы действительно используете сортировку слиянием, мы просим Вас, после каждого осуществленного слияния (то есть, когда соответствующий подмассив уже отсортирован!), выводить индексы граничных элементов и их значения.

Формат входного файла

В первой строке входного файла содержится число n $(1 \le n \le 10^5)$ — число элементов в массиве. Во второй строке находятся n целых чисел, по модулю не превосходящих 10^9.

Формат выходного файла

Выходной файл состоит из нескольких строк.

```
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; }
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.343	41459712	1039245	4403894
1	OK	0.046	12558336	25	105
2	OK	0.031	12419072	6	3
3	OK	0.031	12505088	8	14
4	OK	0.046	12541952	8	14
5	OK	0.046	12587008	42	155
6	OK	0.031	12566528	43	155
7	OK	0.031	12574720	51	179
8	OK	0.046	12509184	45	161
9	OK	0.046	12546048	105	332
10	OK	0.046	12529664	110	343
11	OK	0.046	12562432	107	336
12	OK	0.046	12615680	461	2041
13	OK	0.046	12636160	560	2332
14	OK	0.046	12591104	388	1821
15	OK	0.031	12570624	408	1879
16	OK	0.046	12611584	1042	3776
17	OK	0.046	12619776	1043	3786
18	OK	0.046	12607488	1044	3781
19	OK	0.031	13307904	5587	25512
20	OK	0.031	13373440	6733	28934
21	OK	0.046	13357056	4737	22958
22	OK	0.046	13307904	5685	25796
23	OK	0.046	13484032	10383	39970
24	OK	0.046	13516800	10421	40066
25	OK	0.046	13455360	10420	40049
26	OK	0.078	18010112	65880	305381
27	OK	0.062	18214912	77550	340370
28	OK	0.062	17895424	57488	280210
29	OK	0.046	18038784	68090	311997
30	OK	0.062	17125376	103872	420234
31	OK	0.062	17141760	103940	420401
32	OK	0.062	17174528	103842	420155
33	OK	0.296	38936576	758839	3554240
34	OK	0.312	39989248	875802	3905093
35	OK	0.265	37531648	675241	3303445
36	OK	0.281	38981632	782803	3626101
37	OK	0.343	41418752	1038992	4403366
38	OK	0.296	41459712	1038702	4402359
39	OK	0.296	41377792	1039245	4403894

Число инверсий

1.0 из 1.0 балла (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Инверсией в последовательности чисел A называется такая ситуация, когда $i < j$, а $A_i > A_j$.

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем.

Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией сортировки слиянием.

Формат входного файла

В первой строке входного файла содержится число n $(1 \le n \le 10^5)$ — число элементов в массиве. Во второй строке находятся n целых чисел, по модулю не превосходящих 10^9.

Формат выходного файла

В выходной файл надо вывести число инверсий в массиве.

```
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; }
    }
  }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.125	29990912	1039245	12
1	OK	0.031	11599872	25	4
2	OK	0.015	11313152	6	3
3	OK	0.031	11620352	8	3
4	OK	0.031	11620352	8	3
5	OK	0.031	11603968	42	3
6	OK	0.031	11595776	43	4
7	OK	0.031	11599872	51	3
8	OK	0.031	11628544	45	4
9	OK	0.031	11603968	105	4
10	OK	0.031	11603968	110	4
11	OK	0.031	11665408	107	4
12	OK	0.046	11640832	461	3
13	OK	0.031	11624448	560	6
14	OK	0.031	11612160	388	3
15	OK	0.046	11599872	408	6
16	OK	0.046	11612160	1042	6
17	OK	0.031	11649024	1043	6
18	OK	0.031	11649024	1044	6
19	OK	0.031	11829248	5587	3
20	OK	0.046	11825152	6733	8
21	OK	0.046	11837440	4737	3
22	OK	0.031	11890688	5685	8
23	OK	0.031	11874304	10383	8
24	OK	0.031	11898880	10421	8
25	OK	0.031	11882496	10420	8
26	OK	0.046	12935168	65880	3
27	OK	0.031	12947456	77550	10
28	OK	0.031	12881920	57488	3
29	OK	0.046	12922880	68090	10
30	OK	0.031	13135872	103872	10
31	OK	0.046	13180928	103940	10
32	OK	0.031	13152256	103842	10
33	OK	0.093	29990912	758839	3
34	OK	0.109	29880320	875802	12
35	OK	0.093	28741632	675241	3
36	OK	0.109	28155904	782803	12
37	OK	0.109	29523968	1038992	12
38	OK	0.125	29876224	1038702	12
39	OK	0.093	29843456	1039245	12

Анти-quick sort

1.0 из 1.0 балла (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Хотя QuickSort является очень быстрой сортировкой в среднем, существуют тесты, на которых она работает очень долго. Оценивать время работы алгоритма будем числом сравнений с элементами массива (то есть, суммарным числом сравнений в первом и втором while). Требуется написать программу, генерирующую тест, на котором быстрая сортировка сделает наибольшее число таких сравнений.

Формат входного файла

В первой строке находится единственное число n $(1 \leq n \leq 10^6)$.

Формат выходного файла

Вывести перестановку чисел от 1 до n, на которой быстрая сортировка выполнит максимальное число сравнений. Если таких перестановок несколько, вывести любую из них.

```
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();
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.234	60813312	9	6888897
1	OK	0.046	11300864	3	7
2	OK	0.031	11350016	3	3
3	OK	0.031	11317248	3	5
4	OK	0.015	11358208	3	9
5	OK	0.031	11325440	3	11
6	OK	0.031	11366400	3	13
7	OK	0.031	11358208	3	15
8	OK	0.031	11378688	3	17
9	OK	0.031	11333632	3	19
10	OK	0.031	11309056	4	22
11	OK	0.031	11296768	4	37
12	OK	0.015	11350016	5	293
13	OK	0.015	11403264	6	3894
14	OK	0.031	12087296	7	48901
15	OK	0.031	12070912	7	48895
16	OK	0.093	20680704	8	756196
17	OK	0.078	22994944	8	1556240
18	OK	0.140	35225600	8	3151813
19	OK	0.234	60813312	8	6888889
20	OK	0.234	60788736	9	6888897

K-ая порядковая статистика

1.0 из 1.0 балла (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Дан массив из n элементов. Какие числа являются k_1-ым, (k_1+1)-ым, ..., k_2-ым в порядке неубывания в этом массиве?

Формат входного файла

В первой строке входного файла содержатся три числа: n — размер массива, а также границы интервала k_1 и k_2, при этом $2 \le n \le 4 \cdot 10^7, 1 \le k_1 \le k_2 \le n, k_2 - k_1 < 200$.

Во второй строке находятся числа A, B, C, a_1, a_2, по модулю не превосходящие 10^9. Вы должны получить элементы массива, начиная с третьего, по формуле: $a_i = A \cdot a_{i-2} + B \cdot a_{i-1} + C$. Все вычисления должны производится в 32-битном знаковом типе, переполнения должны игнорироваться.

Формат выходного файла

В первой и единственной строке выходного файла выведите k_1-ое, (k_1+1)-ое, ..., k_2-ое в порядке неубывания числа в массиве a. Числа разделяйте одним пробелом.

Примеры

input.txt	output.txt
5 3 4 2 3 5 1 2	13 48
5 3 4 200000 300000 5 1 2	2 800005

```
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();
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.890	172093440	54	2400
1	OK	0.031	11743232	18	6
2	OK	0.031	11677696	28	9
3	OK	0.062	11710464	32	4
4	OK	0.031	11751424	33	5
5	OK	0.015	11681792	32	10
6	OK	0.031	11755520	33	5
7	OK	0.031	11755520	32	19
8	OK	0.031	11784192	32	21
9	OK	0.031	11751424	25	300
10	OK	0.046	11698176	22	382
11	OK	0.046	11751424	23	477
12	OK	0.031	11694080	35	12
13	OK	0.046	11685888	38	11
14	OK	0.031	11722752	36	1074
15	OK	0.031	11735040	36	561
16	OK	0.031	11796480	37	220
17	OK	0.031	11743232	24	400
18	OK	0.031	11759616	28	1200
19	OK	0.031	11874304	29	1400
20	OK	0.031	11763712	37	12
21	OK	0.031	11804672	45	11
22	OK	0.031	11780096	38	2400
23	OK	0.031	11771904	39	2400
24	OK	0.031	11759616	44	2200
25	OK	0.031	11800576	43	2200
26	OK	0.031	11755520	41	676
27	OK	0.031	12156928	28	600
28	OK	0.031	12136448	31	1400
29	OK	0.031	12234752	32	1600
30	OK	0.031	12132352	37	12
31	OK	0.062	12152832	48	11
32	OK	0.046	12136448	40	2400
33	OK	0.031	12161024	40	2400
34	OK	0.046	12124160	47	2200
35	OK	0.031	12144640	46	2200
36	OK	0.046	12120064	45	200
37	OK	0.062	15745024	32	800
38	OK	0.046	15724544	34	1600

39	OK	0.078	15736832	35	1800
40	OK	0.062	15753216	38	12
41	OK	0.062	15699968	49	11
42	OK	0.062	15745024	40	2400
43	OK	0.062	15806464	40	2003
44	OK	0.062	15724544	49	2200
45	OK	0.062	15757312	47	2200
46	OK	0.078	15691776	48	560
47	OK	1.375	171982848	33	800
48	OK	0.937	171974656	39	2000
49	OK	1.203	171999232	40	2200
50	OK	1.312	172019712	40	12
51	OK	0.875	172015616	52	11
52	OK	1.000	172003328	42	2400
53	OK	1.156	172007424	42	2400
54	OK	1.156	172044288	54	2200
55	OK	1.375	171991040	54	2200
56	OK	1.890	171986944	52	1076
57	OK	1.281	172036096	53	2200
58	OK	1.484	172052480	52	2076
59	OK	1.093	172052480	54	2035
60	OK	1.000	172019712	53	1859
61	OK	1.718	172011520	51	2208
62	OK	1.390	172081152	49	2189
63	OK	1.046	171991040	53	2057
64	OK	1.546	172093440	54	1991
65	OK	1.468	172048384	50	2004
66	OK	1.500	172023808	52	1793
67	OK	1.375	172003328	54	1930

Сортировка пугалом

1.0 из 1.0 балла (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

«Сортировка пугалом» — это давно забытая народная потешка, которую восстановили по летописям специалисты платформы «Открытое образование» специально для этого курса.

Участнику под верхнюю одежду продевают деревянную палку, так что у него оказываются растопырены руки, как у огородного пугала. Перед ним ставятся n матрёшек в ряд. Из-за палки единственное, что он может сделать — это взять в руки две матрешки на расстоянии k друг от друга (то есть i-ую и $(i + k)$-ую), развернуться и поставить их обратно в ряд, таким образом поменяв их местами.

Задача участника — расположить матрёшки по неубыванию размера. Может ли он это сделать?

Формат входного файла

В первой строчке содержатся числа n и k $(1 \le n, k \le 10^5)$ — число матрёшек и размах рук.

Во второй строчке содержится n целых чисел, которые по модулю не превосходят 10^9 — размеры матрёшек.

Формат выходного файла

Выведите «YES», если возможно отсортировать матрёшки по неубыванию размера, и «NO» в противном случае.

```
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();
    }
  }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.109	21995520	1039313	5
1	OK	0.046	11599872	12	4
2	OK	0.031	11640832	16	5
3	OK	0.031	11624448	112	5
4	OK	0.046	11595776	111	4
5	OK	0.031	11583488	112	5
6	OK	0.015	11558912	112	4
7	OK	0.031	11591680	109	5
8	OK	0.031	11575296	112	4
9	OK	0.031	11579392	110	5
10	OK	0.031	11563008	111	4
11	OK	0.015	11620352	108	5
12	OK	0.031	11776000	11674	5
13	OK	0.015	11792384	11707	4
14	OK	0.046	11771904	11712	5
15	OK	0.031	11837440	11754	4
16	OK	0.031	11788288	11708	5
17	OK	0.031	11792384	11740	4
18	OK	0.031	11800576	11726	5
19	OK	0.031	11800576	11680	4
20	OK	0.031	11776000	11741	5
21	OK	0.031	13004800	128736	5
22	OK	0.046	13037568	128832	4
23	OK	0.046	13000704	128751	5
24	OK	0.031	13000704	128866	4
25	OK	0.031	12992512	128700	5
26	OK	0.046	13012992	128707	4
27	OK	0.062	13025280	128729	5
28	OK	0.046	13033472	128807	4
29	OK	0.031	13004800	128784	5
30	OK	0.093	21966848	1039313	5
31	OK	0.109	21995520	1038610	4
32	OK	0.093	21954560	1038875	5
33	OK	0.093	21970944	1038723	4
34	OK	0.062	21950464	1038749	5
35	OK	0.078	21958656	1038747	4
36	OK	0.093	21950464	1039043	5
37	OK	0.062	21966848	1039210	4
38	OK	0.062	21962752	1038967	5