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 №3
неделя третья

Выполнил:
Айгузин Иван Олегович
P3218

Преподаватели:

Романов Алексей Андреевич
Волчек Дмитрий Геннадьевич

Санкт-Петербург

2018

Сортировка целых чисел

2.0 из 2.0 баллов (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	**512 мегабайт**

В этой задаче Вам нужно будет отсортировать много неотрицательных целых чисел.

Вам даны два массива, A и B, содержащие соответственно n и m элементов. Числа, которые нужно будет отсортировать, имеют вид $A_i \cdot B_j$, где $1 \le i \le n$ и $1 \le j \le m$. Иными словами, каждый элемент первого массива нужно умножить на каждый элемент второго массива.

Пусть из этих чисел получится отсортированная последовательность C длиной $n \cdot m$. Выведите сумму каждого десятого элемента этой последовательности (то есть, $C_1 + C_{11} + C_{21} + \ldots$).

Формат входного файла

В первой строке содержатся числа n и m ($1 \le n, m \le 6000$) — размеры массивов. Во второй строке содержится n чисел — элементы массива A. Аналогично, в третьей строке содержится m чисел — элементы массива B. Элементы массива неотрицательны и не превосходят 40000.

Формат выходного файла

Выведите одно число — сумму каждого десятого элемента последовательности, полученной сортировкой попарных произведений элементов массивов A и B.

```
using System;
using System.IO;
using System.Linq;

namespace Task01 {
    public sealed class Program {
        private const int Base = 256;
        private static StreamReader _in;
        private static StreamWriter _out;

        private static void Main() {
            _in = new StreamReader("input.txt");
            _out = new StreamWriter("output.txt");

            Run();

            _in?.Dispose();
            _out?.Dispose();
        }

        private static void Run() {
            var (n, m, arrayA, arrayB) = GetInitValues();

            var array = new int[n * m];
```

```
var max = 0;
var cursor = 0;

for (var i = 0; i < arrayA.Length; i++) {
    var a = arrayA[i];

    for (var j = 0; j < arrayB.Length; j++) {
        var c = a * arrayB[j];
        array[cursor++] = c;

        if (c > max) {
            max = c;
        }
    }
}

var output = new int[array.Length];
var count = new int[Base];

for (var pow = 0; 1L << pow <= max; pow += 8) {
    Array.Clear(count, 0, Base);

    for (var i = 0; i < array.Length; i++) {
        var t = array[i];
        count[(t >> pow) & 255]++;
    }

    for (var i = 1; i < Base; i++) {
        count[i] += count[i - 1];
    }

    for (var i = array.Length - 1; i >= 0; i--) {
        var a = array[i];
        var index = (a >> pow) & 255;
        output[--count[index]] = a;
    }

    for (var i = 0; i < array.Length; i++) {
        array[i] = output[i];
    }
}

long sum = 0;

for (var i = 0; i < array.Length; i += 10) {
    sum += array[i];
}
```

```
            _out?.WriteLine(sum);
        }

        private static (int, int, int[], int[]) GetInitValues() {
            var (n, m) = ReadFirstLine();
            var arrayA = ReadIntList();
            var arrayB = ReadIntList();

            return (n, m, arrayA, arrayB);
        }

        private static (int N, int K) ReadFirstLine() {
            var p = ReadIntList();
            return (p[0], p[1]);
        }

        private static int[] ReadIntList() {
            return _in.ReadLine()
                .Split(' ')
                .Select(int.Parse)
                .ToArray();
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.859	301125632	68699	18
1	OK	0.015	11227136	24	4
2	OK	0.015	11304960	34	3
3	OK	0.046	11288576	38	4
4	OK	0.031	11288576	106	12
5	OK	0.031	11321344	234	13
6	OK	0.031	11309056	698	13
7	OK	0.031	11317248	705	14
8	OK	0.031	11378688	586	14
9	OK	0.015	11943936	34325	14
10	OK	0.046	11476992	5769	14
11	OK	0.031	11431936	3498	14
12	OK	0.046	11333632	924	14
13	OK	0.031	11382784	3494	14
14	OK	0.031	11440128	5772	14
15	OK	0.031	11919360	34449	14
16	OK	0.031	12304384	34368	15
17	OK	0.015	11886592	4006	15
18	OK	0.031	11878400	2886	15
19	OK	0.031	11902976	4009	15
20	OK	0.031	12337152	34361	15
21	OK	0.062	16666624	34966	16
22	OK	0.046	16347136	9167	16
23	OK	0.062	16379904	9162	16
24	OK	0.062	16658432	34917	16
25	OK	0.312	60207104	39991	17
26	OK	0.312	62148608	28668	17
27	OK	0.328	60174336	40034	17
28	OK	0.890	156729344	51489	17
29	OK	0.890	156700672	51525	17
30	OK	1.859	301109248	68655	18
31	OK	1.859	301125632	68625	18
32	OK	1.859	301121536	68699	18

Цифровая сортировка

3.0 из 3.0 баллов (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	**2.5 секунды**
Ограничение по памяти:	256 мегабайт

Дано n строк, выведите их порядок после k фаз цифровой сортировки.

Формат входного файла

В первой строке входного файла содержатся числа n — число строк, m — их длина и k – число фаз цифровой сортировки ($1 \le n \le 10^6, 1 \le k \le m \le 10^6, n \cdot m \le 5 \cdot 10^7$). Далее находится описание строк, **но в нетривиальном формате**. Так, i-ая строка ($1 \le i \le n$) записана в i-ых символах второй, ..., $(m+1)$-ой строк входного файла. Иными словами, строки написаны по вертикали. **Это сделано специально, чтобы сортировка занимала меньше времени.**

Строки состоят из строчных латинских букв: от символа "a" до символа "z" включительно. В таблице символов ASCII все эти буквы располагаются подряд и в алфавитном порядке, код буквы "a" равен 97, код буквы "z" равен 122.

Формат выходного файла

Выведите номера строк в том порядке, в котором они будут после k фаз цифровой сортировки.

```
using System;
using System.IO;
using System.Linq;

namespace Task02 {
    public sealed class Program {
        private const int Base = 26;
        private static StreamReader In;
        private static StreamWriter Out;
        private static readonly int NewLineLength = Environment.NewLine.Length;

        private static void Main(string[] args) {
            if (!args.Contains("console1")) {
                SetupIO();
            }

            Run();
            DisposeIO();
        }

        private static void Run() {
            var (n, len, k, buffer) = GetInitValues();

            // Radix sort
            var indexer = Enumerable.Range(0, n).ToArray();
```

```
            var output = new int[n];
            var count = new int[Base];

            for (var phase = 1; phase <= k; phase++) {
                Array.Clear(count, 0, Base);
                var offset = (len - phase) * n;

                for (var i = 0; i < n; i++) {
                    count[buffer[offset + i]]++;
                }

                for (var i = 1; i < Base; i++) {
                    count[i] += count[i - 1];
                }

                for (var i = n - 1; i >= 0; i--) {
                    var a = buffer[offset + indexer[i]];
                    output[--count[a]] = indexer[i];
                }

                for (var i = 0; i < n; i++) {
                    indexer[i] = output[i];
                }
            }

            //
            for (var i = 0; i < indexer.Length; i++) {
                Out.Write($"{indexer[i] + 1} ");
            }

            Out.WriteLine();
        }

        private static (int n, int m, int k, byte[] buffer) GetInitValues() {
//              var random = new Random(1000);
//              var bytes = Enumerable.Range(0, 200)
//                  .Select(x => (byte) random.Next(0, 25))
//                  .ToArray();
//
//              return (20, 10, 10, bytes);

            var firstLine = In.ReadLine();

            var p = firstLine
                .Split(' ')
                .Select(int.Parse)
                .ToList();
```

```
            var (n, m, k) = (p[0], p[1], p[2]);

            var stream = In.BaseStream;
            stream.Seek(firstLine.Length + NewLineLength, SeekOrigin.Begin);

            var buffer = ReadStrings(n, m, k);
            return (n, m, k, buffer);
        }

        private static byte[] ReadStrings(int n, int m, int k) {
            var stream = In.BaseStream;

            var bufferLength = n * m;
            var buffer = new byte[bufferLength];

            for (var i = 0; i < m; i++) {
                stream.Read(buffer, i * n, n);
                stream.Seek(NewLineLength, SeekOrigin.Current);
            }

            for (var i = 0; i < bufferLength; i++) {
                buffer[i] -= 0x61; // 'a'
            }

            return buffer;
        }

        private static void SetupIO() {
            In = new StreamReader("input.txt");
            Out = new StreamWriter("output.txt");
        }

        private static void DisposeIO() {
            In?.Dispose();
            Out?.Dispose();
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.921	79110144	52000020	6888898
1	OK	0.015	11579392	22	8
2	OK	0.031	11587584	22	8
3	OK	0.031	11608064	22	8
4	OK	0.031	11632640	10	4
5	OK	0.031	11632640	11	6
6	OK	0.031	11661312	130	23
7	OK	0.046	11595776	129	23
8	OK	0.031	11595776	129	23
9	OK	0.031	11595776	129	23
10	OK	0.031	11583488	129	23
11	OK	0.031	11558912	230	53
12	OK	0.031	11632640	229	53
13	OK	0.031	11632640	229	53
14	OK	0.046	11620352	229	53
15	OK	0.015	11587584	229	53
16	OK	0.046	11603968	450	53
17	OK	0.062	11620352	449	53
18	OK	0.031	11558912	450	53
19	OK	0.015	11628544	449	53
20	OK	0.031	11567104	449	53
21	OK	0.046	11616256	530	143
22	OK	0.031	11603968	529	143
23	OK	0.031	11599872	529	143
24	OK	0.046	11591680	529	143
25	OK	0.031	11669504	529	143
26	OK	0.031	11640832	1212	23
27	OK	0.031	11587584	1210	23
28	OK	0.015	11689984	1211	23
29	OK	0.046	11591680	1211	23
30	OK	0.046	11608064	1211	23
31	OK	0.031	11608064	2031	694
32	OK	0.031	11616256	2030	694
33	OK	0.031	11628544	2030	694
34	OK	0.031	11587584	2030	694
35	OK	0.015	11624448	2030	694
36	OK	0.031	11595776	2610	143
37	OK	0.031	11710464	2609	143
38	OK	0.031	11624448	2610	143
39	OK	0.031	11616256	2610	143
40	OK	0.031	11616256	2609	143
41	OK	0.031	11595776	4051	694
42	OK	0.046	11612160	4050	694
43	OK	0.031	11599872	4051	694
44	OK	0.031	11587584	4051	694
45	OK	0.031	11649024	4051	694
46	OK	0.031	11612160	6012	23
47	OK	0.046	11640832	6010	23
48	OK	0.031	11587584	6012	23
49	OK	0.031	11620352	6012	23
50	OK	0.031	11653120	6010	23
51	OK	0.031	11636736	10213	294
52	OK	0.015	11616256	10211	294
53	OK	0.031	11591680	10212	294
54	OK	0.046	11649024	10212	294
55	OK	0.031	11620352	10212	294
56	OK	0.031	11698176	20052	3895

57	OK	0.031	11730944	20051	3895
58	OK	0.046	11702272	20052	3895
59	OK	0.046	11776000	20052	3895
60	OK	0.046	11702272	20051	3895
61	OK	0.031	11685888	26012	143
62	OK	0.015	11620352	26010	143
63	OK	0.031	11644928	26012	143
64	OK	0.031	11603968	26011	143
65	OK	0.031	11608064	26012	143
66	OK	0.031	11653120	40413	694
67	OK	0.031	11685888	40411	694
68	OK	0.031	11673600	40413	694
69	OK	0.031	11653120	40412	694
70	OK	0.031	11661312	40413	694
71	OK	0.046	11628544	52014	143
72	OK	0.031	11771904	52011	143
73	OK	0.046	11735040	52013	143
74	OK	0.031	11673600	52013	143
75	OK	0.031	11657216	52013	143
76	OK	0.046	11739136	102015	294
77	OK	0.031	11726848	102012	294
78	OK	0.031	11702272	102014	294
79	OK	0.031	11718656	102014	294
80	OK	0.031	11722752	102014	294
81	OK	0.031	13434880	200033	108896
82	OK	0.031	13471744	200032	108896
83	OK	0.046	13422592	200032	108896
84	OK	0.046	13418496	200032	108896
85	OK	0.031	13443072	200032	108896
86	OK	0.031	12505088	250112	23895
87	OK	0.031	12472320	250111	23895
88	OK	0.031	12533760	250112	23895
89	OK	0.031	12488704	250111	23895
90	OK	0.031	12439552	250112	23895
91	OK	0.046	13639680	400053	108896
92	OK	0.031	13602816	400052	108896
93	OK	0.031	13660160	400053	108896
94	OK	0.046	13639680	400053	108896
95	OK	0.046	13623296	400053	108896
96	OK	0.031	12193792	501014	3895
97	OK	0.046	12218368	501012	3895
98	OK	0.046	12251136	501014	3895
99	OK	0.031	12242944	501014	3895
100	OK	0.031	12214272	501013	3895
101	OK	0.046	13225984	1000414	23895
102	OK	0.031	13217792	1000412	23895
103	OK	0.031	13316096	1000414	23895
104	OK	0.062	13209600	1000413	23895
105	OK	0.046	13238272	1000414	23895
106	OK	0.281	13606912	2400018	23
107	OK	0.250	13602816	2400013	23
108	OK	0.312	13606912	2400018	23
109	OK	0.296	13590528	2400018	23
110	OK	0.296	13586432	2400018	23
111	OK	0.078	16076800	2500113	288896
112	OK	0.078	16060416	2500112	288896
113	OK	0.078	16052224	2500113	288896
114	OK	0.062	16089088	2500112	288896

115	OK	0.078	16035840	2500113	288896
116	OK	0.078	15863808	4004016	8895
117	OK	0.046	15855616	4004013	8895
118	OK	0.093	15831040	4004016	8895
119	OK	0.046	15814656	4004015	8895
120	OK	0.078	15802368	4004016	8895
121	OK	0.093	18653184	5000215	288896
122	OK	0.046	18780160	5000213	288896
123	OK	0.109	18784256	5000214	288896
124	OK	0.078	18669568	5000214	288896
125	OK	0.093	18690048	5000214	288896
126	OK	0.187	24543232	10000216	588897
127	OK	0.093	24592384	10000214	588897
128	OK	0.187	24567808	10000215	588897
129	OK	0.109	24551424	10000215	588897
130	OK	0.109	24539136	10000215	588897
131	OK	0.468	36331520	20000216	1288897
132	OK	0.140	36286464	20000214	1288897
133	OK	0.375	36290560	20000215	1288897
134	OK	0.250	36306944	20000215	1288897
135	OK	0.281	36323328	20000215	1288897
136	OK	0.343	38690816	25001015	288896
137	OK	0.093	38715392	25001013	288896
138	OK	0.328	38768640	25001015	288896
139	OK	0.234	38739968	25001015	288896
140	OK	0.187	38756352	25001015	288896
141	OK	0.937	36614144	26000018	143
142	OK	0.671	36651008	26000013	143
143	OK	0.984	36597760	26000018	143
144	OK	0.937	36589568	26000018	143
145	OK	0.843	36614144	26000018	143
146	OK	0.406	36679680	25100017	1894
147	OK	0.171	36651008	25100013	1894
148	OK	0.390	36642816	25100017	1894
149	OK	0.296	36683776	25100017	1894
150	OK	0.234	36663296	25100016	1894
151	OK	0.343	37183488	25010016	23895
152	OK	0.109	37203968	25010013	23895
153	OK	0.328	37179392	25010016	23895
154	OK	0.140	37183488	25010015	23895
155	OK	0.281	37294080	25010016	23895
156	OK	0.671	45940736	25000114	3388897
157	OK	0.250	46010368	25000113	3388897
158	OK	0.578	45948928	25000114	3388897
159	OK	0.343	45973504	25000114	3388897
160	OK	0.234	45998080	25000113	3388897
161	OK	0.531	51855360	40040018	8895
162	OK	0.187	51888128	40040014	8895
163	OK	0.546	51855360	40040018	8895
164	OK	0.312	51855360	40040017	8895
165	OK	0.468	51859456	40040018	8895
166	OK	0.796	51621888	40400019	694
167	OK	0.359	51617792	40400014	694
168	OK	0.828	51679232	40400019	694
169	OK	0.515	51650560	40400018	694
170	OK	0.375	51646464	40400016	694
171	OK	0.531	53407744	40004017	108896
172	OK	0.125	53387264	40004014	108896
173	OK	0.484	53415936	40004017	108896

174	OK	0.265	53440512	40004016	108896
175	OK	0.328	53415936	40004017	108896
176	OK	0.656	56299520	40000416	1288897
177	OK	0.187	56311808	40000414	1288897
178	OK	0.687	56266752	40000416	1288897
179	OK	0.234	56283136	40000415	1288897
180	OK	0.187	56262656	40000414	1288897
181	OK	1.328	61612032	51000019	294
182	OK	0.750	61628416	51000014	294
183	OK	1.281	61575168	51000019	294
184	OK	0.765	61616128	51000018	294
185	OK	1.140	61632512	51000019	294
186	OK	0.703	61743104	50100018	3895
187	OK	0.234	61734912	50100014	3895
188	OK	0.718	61730816	50100018	3895
189	OK	0.375	61673472	50100018	3895
190	OK	0.484	61714432	50100018	3895
191	OK	1.484	79048704	50000115	6888898
192	OK	0.437	79110144	50000114	6888898
193	OK	1.406	79101952	50000115	6888898
194	OK	1.046	79060992	50000115	6888898
195	OK	1.234	79110144	50000115	6888898
196	OK	0.765	61648896	50200019	1894
197	OK	0.296	61677568	50200014	1894
198	OK	0.765	61648896	50200018	1894
199	OK	0.562	61714432	50200018	1894
200	OK	0.656	61648896	50200018	1894
201	OK	0.718	64598016	50001016	588897
202	OK	0.171	64557056	50001014	588897
203	OK	0.687	64622592	50001016	588897
204	OK	0.468	64614400	50001016	588897
205	OK	0.234	64675840	50001015	588897
206	OK	0.609	63713280	50002017	288896
207	OK	0.156	63692800	50002014	288896
208	OK	0.640	63664128	50002016	288896
209	OK	0.390	63676416	50002016	288896
210	OK	0.625	63705088	50002016	288896
211	OK	1.062	70934528	50000216	3388897
212	OK	0.281	70950912	50000214	3388897
213	OK	1.171	70942720	50000215	3388897
214	OK	0.859	71041024	50000215	3388897
215	OK	0.750	70934528	50000215	3388897
216	OK	1.843	61624320	52000020	143
217	OK	1.281	61591552	52000014	143
218	OK	1.921	61620224	52000019	143
219	OK	1.859	61636608	52000019	143
220	OK	1.328	61648896	52000018	143
221	OK	0.656	62894080	50010017	48896
222	OK	0.140	62877696	50010014	48896
223	OK	0.625	62869504	50010017	48896
224	OK	0.296	62898176	50010017	48896
225	OK	0.265	62894080	50010017	48896
226	OK	0.640	62214144	50020018	23895
227	OK	0.203	62263296	50020014	23895
228	OK	0.625	62218240	50020017	23895
229	OK	0.546	62181376	50020017	23895
230	OK	0.546	62222336	50020017	23895