Министерство образования и науки Российской Федерации

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

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Алгоритмы и структуры данных»

ОТЧЁТ

по лабораторной работе №3 (Week 3 Openedu)

Студент Луговских Савелий P3218

Преподаватель Муромцев Дмитрий Ильич

Санкт-Петербург

2019 г.

Содержание

Задача 1 Сортировка целых чисел

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 Ai * Bj, где $1 \leq i \leq n$ и $1 \leq j \leq m$ Иными словами, каждый элемент первого массива нужно умножить на каждый элемент второго массива.

Пусть из этих чисел получится отсортированная последовательность C длиной n * m. Выведите сумму каждого десятого элемента этой последовательности.

Формат входного файла

В первой строке содержатся числа n и m ($1 \leq n, m \leq 6000$) — размеры массивов. Во второй строке содержится чисел — элементы массива A. Аналогично, в третьей строке содержится 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 произведенй элементов массивов A и B.

Примеры

input.txt	output.txt
4 4 7 1 4 9 2 7 8 11	51

Исходный код к задаче 1

```
namespace
Third_week
            {
                using System.IO;
                using System.Linq;

                class Program
```

```
    {
        static void Main()
        {
            StreamReader input = new StreamReader("input.txt");
            StreamWriter output = new StreamWriter("output.txt");
            input.ReadLine();
            string[] tokens = input.ReadLine().Split(' ');
            int[] firstArray = new int[tokens.Count()];
            for (int i = 0; i < tokens.Count(); i++)
            {
                firstArray[i] = int.Parse(tokens[i]);
            }

            tokens = input.ReadLine().Split(' ');
            int[] secondArray = new int[tokens.Count()];
            for (int i = 0; i < tokens.Count(); i++)
            {
                secondArray[i] = int.Parse(tokens[i]);
            }

            input.Close();
            (byte, byte, byte, byte)[] byteArray = new (byte, byte, byte,
byte)[firstArray.Length * secondArray.Length * 2];
            for (int i = 0; i < firstArray.Length; i++)
            {
                for (int j = 0; j < secondArray.Length; j++)
                {
                    byteArray[(i * secondArray.Length) + j].Item1 =
(byte)((firstArray[i] * secondArray[j]) & 255);
                    byteArray[(i * secondArray.Length) + j].Item2 =
(byte)((firstArray[i] * secondArray[j]) >> 8 & 255);
                    byteArray[(i * secondArray.Length) + j].Item3 =
                        (byte)((firstArray[i] * secondArray[j]) >> 16 & 255);
                    byteArray[(i * secondArray.Length) + j].Item4 =
                        (byte)((firstArray[i] * secondArray[j]) >> 24 & 255);
                }
            }

            int[] countNumbers = new int[1024];
            for (int i = 0; i < byteArray.Length / 2; i++)
            {
                countNumbers[byteArray[i].Item1]++;
                countNumbers[byteArray[i].Item2 + 256]++;
                countNumbers[byteArray[i].Item3 + 512]++;
                countNumbers[byteArray[i].Item4 + 768]++;
            }

            for (int i = 1; i < 256; i++)
```

```
            {
                countNumbers[i] += countNumbers[i - 1];
                countNumbers[i + 256] += countNumbers[i + 255];
                countNumbers[i + 512] += countNumbers[i + 511];
                countNumbers[i + 768] += countNumbers[i + 767];
            }

            for (int j = byteArray.Length / 2 - 1; j > -1; j--)
            {
                byteArray[countNumbers[byteArray[j].Item1] - 1 +
byteArray.Length / 2] = byteArray[j];
                countNumbers[byteArray[j].Item1]--;
            }

            for (int j = byteArray.Length - 1; j > byteArray.Length / 2 - 1;
j--)
            {
                byteArray[countNumbers[byteArray[j].Item2 + 256] - 1] =
byteArray[j];
                countNumbers[byteArray[j].Item2 + 256]--;
            }

            for (int j = byteArray.Length / 2 - 1; j > -1; j--)
            {
                byteArray[countNumbers[byteArray[j].Item3 + 512] - 1 +
byteArray.Length / 2] = byteArray[j];
                countNumbers[byteArray[j].Item3 + 512]--;
            }

            for (int j = byteArray.Length - 1; j > byteArray.Length / 2 - 1;
j--)
            {
                byteArray[countNumbers[byteArray[j].Item4 + 768] - 1] =
byteArray[j];
                countNumbers[byteArray[j].Item4 + 768]--;
            }

            long sum = 0;
            for (int i = 0; i < byteArray.Length / 2; i += 10)
            {
                sum += (byteArray[i].Item4 << 24) + (byteArray[i].Item3 << 16)
+ (byteArray[i].Item2 << 8)
                        + byteArray[i].Item1;
            }

            output.WriteLine(sum);
            output.Close();
        }
```

```
                }
            }
```

Бенчмарк к задаче 1

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.781	289935360	68699	16
1	OK	0.000	2224128	24	2
2	OK	0.000	2220032	34	1
3	OK	0.015	2224128	38	2
4	OK	0.000	2240512	106	10
5	OK	0.000	2220032	234	11
6	OK	0.000	2240512	698	11
7	OK	0.000	2228224	705	12
8	OK	0.015	2252800	586	12
9	OK	0.000	2215936	34325	12
10	OK	0.015	2273280	5769	12
11	OK	0.000	2273280	3498	12
12	OK	0.031	2240512	924	12
13	OK	0.000	2269184	3494	12
14	OK	0.015	2285568	5772	12
15	OK	0.000	2220032	34449	12
16	OK	0.000	2351104	34368	13
17	OK	0.000	2293760	4006	13
18	OK	0.000	2314240	2886	13
19	OK	0.000	2293760	4009	13
20	OK	0.000	2351104	34361	13
21	OK	0.031	6672384	34966	14
22	OK	0.015	6623232	9167	14
23	OK	0.046	6623232	9162	14
24	OK	0.031	6668288	34917	14
25	OK	0.296	49881088	39991	15
26	OK	0.312	51863552	28668	15
27	OK	0.296	49881088	40034	15
28	OK	0.890	145903616	51489	15
29	OK	0.890	145907712	51525	15
30	OK	1.765	289935360	68655	16
31	OK	1.765	289931264	68625	16
32	OK	1.781	289931264	68699	16

Задача 2 Цифровая сортировка

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 ($1 \le n \le 10^6, 1 \le k \le m \le 10^6, n * m \le 5 * 10^7$). 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 kфаз цифровой сортировки.

Примеры

input.txt	output.txt
3 3 1 bab bba baa	2 3 1
3 3 2 bab bba baa	3 2 1
3 3 3 bab bba baa	2 3 1

Исходный код к задаче 2

```
namespace
Digital_sorting
                    {
                        using System.IO;
                        using System.IO.MemoryMappedFiles;

                        class Program
                        {
                            static void Main()
                            {

                               StreamReader inputAdd = new StreamReader("input.txt");
                                StreamWriter output = new StreamWriter("output.txt");
                                string[] tokens = inputAdd.ReadLine().Split(' ');
                                int strLength = int.Parse(tokens[0]), strCount =
                    int.Parse(tokens[1]), maxSteps = int.Parse(tokens[2]);
                                inputAdd.Close();

                                var input = MemoryMappedFile.CreateFromFile(
                                    "input.txt",
                                    System.IO.FileMode.Open,
                                    "fileHandle",
                                    4096 * 4096);
                                var myAccessor = input.CreateViewStream();

                                byte[] allBytes = new byte[tokens[0].Length +
                    tokens[1].Length + tokens[2].Length + 4 + strCount * (strLength + 2)];
                                for (int i = 0; i < allBytes.Length; i++)
                                {
                                    allBytes[i] = (byte)myAccessor.ReadByte();
                                }

                                myAccessor.Close();
                                int[] positions = new int[strLength * 2];
                                int[] countNumbers = new int[26 * maxSteps];
                                for (int i = 0; i < strLength; i++)
                                {
                                    positions[i] = i;
                                }

                                for (int y = 0; y < maxSteps; y++)
                                {
                                    for (int x = 0; x < strLength; x++)
                                    {
                                        countNumbers[allBytes[positions[x + (strLength * (y %
                    2))] + allBytes.Length - ((y + 1) * strLength)
```

```
- (y + 1) * 2] - 97
                                    + 26 * y]++;
                }

                for (int i = 1; i < 26; i++)
                {
                    countNumbers[i + (26 * y)] += countNumbers[i - 1 +
(26 * y)];
                }

                for (int x = strLength - 1; x > -1; x--)
                {
                    positions[countNumbers[allBytes[positions[x +
(strLength * (y % 2))] + allBytes.Length
                                                        - ((y + 1) *
strLength) - (y + 1) * 2] - 97 + (26 * y)]-- - 1
                                + strLength * ((y + 1) % 2)] = positions[x
+ strLength * (y % 2)];
                }
            }

            for (int i = 0; i < strLength; i++)
            {
                output.Write(positions[i + (maxSteps % 2 * strLength)] +
1 + " ");
            }

            output.Close();
        }
    }
}
```

Бенчмарк к задаче 2

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.843	227307520	52000020	6888896
1	OK	0.031	11038720	22	6
2	OK	0.015	11038720	22	6
3	OK	0.031	10989568	22	6
4	OK	0.062	11010048	10	2
5	OK	0.031	10989568	11	4
6	OK	0.031	11046912	130	21
7	OK	0.031	11005952	129	21
8	OK	0.046	10997760	129	21

9	OK	0.015	11042816	129	21
10	OK	0.031	11014144	129	21
11	OK	0.031	11055104	230	51
12	OK	0.031	11018240	229	51
13	OK	0.031	11026432	229	51
14	OK	0.046	11010048	229	51
15	OK	0.031	11022336	229	51
16	OK	0.015	11042816	450	51
17	OK	0.062	11026432	449	51
18	OK	0.031	11124736	450	51
19	OK	0.031	11014144	449	51
20	OK	0.031	11010048	449	51
21	OK	0.015	11038720	530	141
22	OK	0.046	10989568	529	141
23	OK	0.031	11022336	529	141
24	OK	0.015	11034624	529	141
25	OK	0.031	11014144	529	141
26	OK	0.046	11046912	1212	21
27	OK	0.031	11067392	1210	21
28	OK	0.031	11067392	1211	21
29	OK	0.031	11014144	1211	21
30	OK	0.031	11051008	1211	21
31	OK	0.031	11030528	2031	692
32	OK	0.031	11030528	2030	692
33	OK	0.031	11034624	2030	692
34	OK	0.031	11063296	2030	692
35	OK	0.031	11087872	2030	692
36	OK	0.031	11034624	2610	141
37	OK	0.031	11059200	2609	141
38	OK	0.031	11038720	2610	141
39	OK	0.031	11046912	2610	141
40	OK	0.031	11030528	2609	141
41	OK	0.031	11055104	4051	692
42	OK	0.062	11079680	4050	692
43	OK	0.046	11075584	4051	692
44	OK	0.046	11046912	4051	692
45	OK	0.031	11059200	4051	692
46	OK	0.046	11075584	6012	21
47	OK	0.031	11096064	6010	21

48	OK	0.031	11087872	6012	21
49	OK	0.015	11063296	6012	21
50	OK	0.031	11038720	6010	21
51	OK	0.031	11145216	10213	292
52	OK	0.031	11079680	10211	292
53	OK	0.031	11083776	10212	292
54	OK	0.046	11124736	10212	292
55	OK	0.015	11059200	10212	292
56	OK	0.031	11288576	20052	3893
57	OK	0.031	11223040	20051	3893
58	OK	0.046	11284480	20052	3893
59	OK	0.031	11309056	20052	3893
60	OK	0.031	11239424	20051	3893
61	OK	0.015	11206656	26012	141
62	OK	0.031	11108352	26010	141
63	OK	0.015	11243520	26012	141
64	OK	0.031	11104256	26011	141
65	OK	0.015	11214848	26012	141
66	OK	0.046	11358208	40413	692
67	OK	0.031	11128832	40411	692
68	OK	0.031	11309056	40413	692
69	OK	0.031	11300864	40412	692
70	OK	0.046	11280384	40413	692
71	OK	0.031	11399168	52014	141
72	OK	0.078	11227136	52011	141
73	OK	0.046	11407360	52013	141
74	OK	0.031	11300864	52013	141
75	OK	0.031	11354112	52013	141
76	OK	0.015	11767808	102015	292
77	OK	0.015	11366400	102012	292
78	OK	0.031	11800576	102014	292
79	OK	0.031	11673600	102014	292
80	OK	0.031	11513856	102014	292
81	OK	0.046	15003648	200033	108894
82	OK	0.078	13246464	200032	108894
83	OK	0.031	14913536	200032	108894
84	OK	0.046	14405632	200032	108894
85	OK	0.093	14684160	200032	108894
86	OK	0.062	13357056	250112	23893

87	OK	0.031	12333056	250111	23893
88	OK	0.046	13291520	250112	23893
89	OK	0.031	12431360	250111	23893
90	OK	0.046	12623872	250112	23893
91	OK	0.046	16171008	400053	108894
92	OK	0.031	14622720	400052	108894
93	OK	0.046	16179200	400053	108894
94	OK	0.046	15532032	400053	108894
95	OK	0.046	16171008	400053	108894
96	OK	0.031	15024128	501014	3893
97	OK	0.031	12988416	501012	3893
98	OK	0.046	15011840	501014	3893
99	OK	0.031	13578240	501014	3893
100	OK	0.031	13217792	501013	3893
101	OK	0.046	17838080	1000414	23893
102	OK	0.031	16039936	1000412	23893
103	OK	0.046	17838080	1000414	23893
104	OK	0.046	17108992	1000413	23893
105	OK	0.046	17850368	1000414	23893
106	OK	0.156	32464896	2400018	21
107	OK	0.078	25534464	2400013	21
108	OK	0.156	32464896	2400018	21
109	OK	0.140	32468992	2400018	21
110	OK	0.140	32460800	2400018	21
111	OK	0.078	28012544	2500113	288894
112	OK	0.062	19664896	2500112	288894
113	OK	0.093	27992064	2500113	288894
114	OK	0.062	20905984	2500112	288894
115	OK	0.093	27234304	2500113	288894
116	OK	0.093	25272320	4004016	8893
117	OK	0.062	23007232	4004013	8893
118	OK	0.109	25255936	4004016	8893
119	OK	0.062	23433216	4004015	8893
120	OK	0.093	25239552	4004016	8893
121	OK	0.140	40296448	5000215	288894
122	OK	0.078	24408064	5000213	288894
123	OK	0.140	40071168	5000214	288894
124	OK	0.078	28397568	5000214	288894
125	OK	0.125	38858752	5000214	288894

126	OK	0.265	77848576	10000216	588895
127	OK	0.125	38281216	10000214	588895
128	OK	0.234	77479936	10000215	588895
129	OK	0.156	43094016	10000215	588895
130	OK	0.156	45543424	10000215	588895
131	OK	0.484	107216896	20000216	1288895
132	OK	0.234	60186624	20000214	1288895
133	OK	0.546	107270144	20000215	1288895
134	OK	0.375	85487616	20000215	1288895
135	OK	0.468	87146496	20000215	1288895
136	OK	0.500	148332544	25001015	288894
137	OK	0.203	63762432	25001013	288894
138	OK	0.484	147750912	25001015	288894
139	OK	0.375	108130304	25001015	288894
140	OK	0.312	85004288	25001015	288894
141	OK	0.703	103538688	26000018	141
142	OK	0.312	91901952	26000013	141
143	OK	0.734	103428096	26000018	141
144	OK	0.671	103456768	26000018	141
145	OK	0.578	103473152	26000018	141
146	OK	0.437	93556736	25100017	1892
147	OK	0.234	88559616	25100013	1892
148	OK	0.421	93548544	25100017	1892
149	OK	0.343	93564928	25100017	1892
150	OK	0.265	93507584	25100016	1892
151	OK	0.406	79294464	25010016	23893
152	OK	0.203	75681792	25010013	23893
153	OK	0.406	79261696	25010016	23893
154	OK	0.250	81301504	25010015	23893
155	OK	0.375	79233024	25010016	23893
156	OK	0.875	124608512	25000114	3388895
157	OK	0.375	72560640	25000113	3388895
158	OK	0.859	124637184	25000114	3388895
159	OK	0.515	106561536	25000114	3388895
160	OK	0.359	74510336	25000113	3388895
161	OK	0.640	110907392	40040018	8893
162	OK	0.328	105033728	40040014	8893
163	OK	0.656	104857600	40040018	8893
164	OK	0.484	108949504	40040017	8893

165	OK	0.578	109142016	40040018	8893
166	OK	0.843	138887168	40400019	692
167	OK	0.359	129163264	40400014	692
168	OK	0.765	138944512	40400019	692
169	OK	0.531	135720960	40400018	692
170	OK	0.375	129609728	40400016	692
171	OK	0.640	116998144	40004017	108894
172	OK	0.328	111824896	40004014	108894
173	OK	0.640	117043200	40004017	108894
174	OK	0.453	118202368	40004016	108894
175	OK	0.500	115830784	40004017	108894
176	OK	0.921	226693120	40000416	1288895
177	OK	0.359	100720640	40000414	1288895
178	OK	1.062	226717696	40000416	1288895
179	OK	0.437	121851904	40000415	1288895
180	OK	0.375	103985152	40000414	1288895
181	OK	1.171	168366080	51000019	292
182	OK	0.515	159440896	51000014	292
183	OK	1.156	168394752	51000019	292
184	OK	0.640	169013248	51000018	292
185	OK	0.953	168390656	51000019	292
186	OK	0.796	149831680	50100018	3893
187	OK	0.453	148402176	50100014	3893
188	OK	0.828	150335488	50100018	3893
189	OK	0.531	150433792	50100018	3893
190	OK	0.640	151302144	50100018	3893
191	OK	1.843	227266560	50000115	6888896
192	OK	0.703	127737856	50000114	6888896
193	OK	1.796	227295232	50000115	6888896
194	OK	1.343	227270656	50000115	6888896
195	OK	1.625	227307520	50000115	6888896
196	OK	0.843	180264960	50200019	1892
197	OK	0.437	175165440	50200014	1892
198	OK	0.859	180260864	50200018	1892
199	OK	0.671	180252672	50200018	1892
200	OK	0.750	180269056	50200018	1892
201	OK	1.000	223670272	50001016	588895
202	OK	0.390	116940800	50001014	588895
203	OK	1.062	223703040	50001016	588895

204	OK	0.656	223670272	50001016	588895
205	OK	0.453	139223040	50001015	588895
206	OK	0.921	225116160	50002017	288894
207	OK	0.406	114577408	50002014	288894
208	OK	0.859	223170560	50002016	288894
209	OK	0.640	217382912	50002016	288894
210	OK	0.859	223133696	50002016	288894
211	OK	1.578	223006720	50000216	3388895
212	OK	0.515	123207680	50000214	3388895
213	OK	1.484	223940608	50000215	3388895
214	OK	1.328	223920128	50000215	3388895
215	OK	1.093	223870976	50000215	3388895
216	OK	1.468	178335744	52000020	141
217	OK	0.671	172724224	52000014	141
218	OK	1.390	179548160	52000019	141
219	OK	1.328	179494912	52000019	141
220	OK	0.718	178278400	52000018	141
221	OK	0.781	139071488	50010017	48894
222	OK	0.390	135553024	50010014	48894
223	OK	0.781	141725696	50010017	48894
224	OK	0.515	141778944	50010017	48894
225	OK	0.484	139190272	50010017	48894
226	OK	0.796	135647232	50020018	23893
227	OK	0.375	130281472	50020014	23893
228	OK	0.765	135892992	50020017	23893
229	OK	0.765	137142272	50020017	23893
230	OK	0.703	135958528	50020017	23893