Санкт-Петербургский Научный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Задачи второй недели

по курсу «Алгоритмы и структуры данных»

на Openedu

Выполнил: студент группы P3218

Артамонов Александр Владимирович

Санкт-Петербург, 2019г

Задача 1. Сортировка слиянием

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 ($1 \le n \le 105$) — число элементов в массиве. Во второй строке находятся n целых чисел, по модулю не превосходящих 109.

Формат выходного файла

Выходной файл состоит из нескольких строк.

В **последней строке** выходного файла требуется вывести отсортированный в порядке неубывания массив, данный на входе. Между любыми двумя числами должен стоять ровно один пробел.

Все предшествующие строки описывают осуществленные слияния, по одному на каждой строке. Каждая такая строка должна содержать по четыре числа: If Il Vf Vl, где If — индекс начала области слияния, Il — индекс конца области слияния, Vf — значение первого элемента области слияния, Vl — значение последнего элемента области слияния.

Все индексы начинаются с единицы (то есть, $1 \le If \le Il \le n$). **Индексы области слияния должны описывать положение области слияния в исходном массиве!** Допускается не выводить информацию о слиянии для подмассива длиной 1, так как он отсортирован по определению.

Приведем небольшой пример: отсортируем массив [9, 7, 5, 8]. Рекурсивная часть сортировки слиянием (процедура SORT(A, L, R), где A — сортируемый массив, L — индекс начала области слияния, R — индекс конца области слияния) будет вызвана с A=[9,7,5,8], L=1, R=4 и выполнит следующие действия:

- разделит область слияния [1;4] на две части, [1;2] и [3;4];
- выполнит вызов SORT(A, L=1, R=2):
    - разделит область слияния [1;2] на две части, [1;1] и [2;2];
    - получившиеся части имеют единичный размер, рекурсивные вызовы можно не делать;
    - осуществит слияние, после чего A станет равным [7,9,5,8];

        - выведет описание слияния: If=L=1, Il=R=2, Vf=AL=7, Vl=AR=9.
    - выполнит вызов SORT(A, L=3, R=4):
        - разделит область слияния [3;4] на две части, [3;3] и [4;4];
        - получившиеся части имеют единичный размер, рекурсивные вызовы можно не делать;
        - осуществит слияние, после чего A станет равным [7,9,5,8];
        - выведет описание слияния: If=L=3, Il=R=4, Vf=AL=5, Vl=AR=8.
    - осуществит слияние, после чего A станет равным [5,7,8,9];
    - выведет описание слияния: If=L=1, Il=R=4, Vf=AL=5, Vl=AR=9.

Описания слияний могут идти в произвольном порядке, необязательно совпадающем с порядком их выполнения. Однако, с целью повышения производительности, рекомендуем выводить эти описания сразу, не храня их в памяти. Именно по этой причине отсортированный массив выводится в самом конце.

Пример

input.txt	output.txt
10 1 8 2 1 4 7 3 2 3 6	1 2 1 8 3 4 1 2 1 4 1 8 5 6 4 7 1 6 1 8 7 8 2 3 9 10 3 6 7 10 2 6 1 10 1 8 1 1 2 2 3 3 4 6 7 8

Код программы (C++)

```
#include <fstream>
using namespace std;


//"Сливает" два отсортированных по возрастанию массива в один отсортированный по возрастанию
long* merge_arrays(long* array1, int length1, long* array2, int length2, int leftborder, ofstream& output) {
	int result_length = length1 + length2;
	long* result_array = new long[result_length];

	int i = 0;
	int j = 0;

	for (int counter = 0; counter < result_length; counter++) {
		if ((array1[i] <= array2[j] || j == length2) && i != length1) { //Если элемент "левого" массива меньше элемента "правого" или правый закончился,
			result_array[counter] = array1[i];
			//и при условии, что не закончился левый, элемент левого массива ставится на следующую позицию
			i++;
				//результирующего массива
		}
		else {
				//В обратном случае в результирующий массив добавляется элемент правого массива
			result_array[counter] = array2[j];
			j++;
		}
	}
	output << leftborder << " " << leftborder + result_length - 1 << " " << result_array[0] << " " << result_array[result_length - 1] << "\n";
	return result_array;
}

//Сортирует массив определённой длины, сохраняя позицию его первого элемента в исходном массиве
long* sort_array(long* array, int length, int leftborder, ofstream &output) {
	if (length == 1) {
		return array;
	}

	//Деление массива на два подмассива, где первый <= второго
	int left_array_size, right_array_size;
	if (length % 2 == 1) {
		left_array_size = length / 2;
		right_array_size = length / 2 + 1;
	}
	else {
		left_array_size = length / 2;
		right_array_size = length / 2;
	}
	long* left_array = new long[left_array_size];
	long* right_array = new long[right_array_size];

	for (int i = 0; i < length; i++) {
		if (i < left_array_size) {
			left_array[i] = array[i];
		}
		else {
			right_array[i - left_array_size] = array[i];
		}
```

```
	}

	//Вызов функции для каждого из двух подмассивов для их сортировки
	left_array = sort_array(left_array, left_array_size, leftborder, output);
	right_array = sort_array(right_array, right_array_size, leftborder +
left_array_size, output);

	//Слияние отсортированных массивов в один
	return merge_arrays(left_array, left_array_size, right_array, right_array_size,
leftborder, output);
}

long* read_array_from_file(string file_name, int &size) {
	ifstream input(file_name);
	input >> size;
	long* array = new long[size];

	for (int i = 0; i < size; i++) {
		input >> array[i];
	}

	input.close();
	return array;
}

int main() {

	int size;

	long* array = read_array_from_file("input.txt", size);

	ofstream output("output.txt");

	array = sort_array(array, size, 1, output);

	for (int i = 0; i < size; i++) {
		output << array[i];
		if (i != size - 1) output << " ";
	}

	output.close();
	return 0;
}
```

Бенчмарк (задача 1)

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.578	20627456	1039245	4403712
1	OK	0.000	2588672	25	104
2	OK	0.000	2576384	6	1
3	OK	0.000	2576384	8	12
4	OK	0.000	2355200	8	12
5	OK	0.015	2351104	42	153
6	OK	0.000	2367488	43	153
7	OK	0.000	2367488	51	177
8	OK	0.015	2351104	45	160
9	OK	0.015	2355200	105	329
10	OK	0.000	2355200	110	342
11	OK	0.000	2355200	107	335
12	OK	0.015	2383872	461	2043
13	OK	0.000	2387968	560	2330
14	OK	0.000	2383872	388	1821
15	OK	0.000	2396160	408	1882
16	OK	0.015	2371584	1042	3775
17	OK	0.015	2383872	1043	3783
18	OK	0.000	2383872	1044	3774
19	OK	0.000	2527232	5587	25512
20	OK	0.015	2510848	6733	28936
21	OK	0.000	2510848	4737	22959
22	OK	0.015	2514944	5685	25798
23	OK	0.015	2514944	10383	39967
24	OK	0.015	2514944	10421	40059
25	OK	0.015	2514944	10420	40056
26	OK	0.046	4046848	65880	305387
27	OK	0.062	4046848	77550	340375
28	OK	0.046	4046848	57488	280212
29	OK	0.046	4046848	68090	311996
30	OK	0.062	4042752	103872	420188
31	OK	0.062	4046848	103940	420365
32	OK	0.062	4038656	103842	420121
33	OK	0.531	20615168	758839	3554250
34	OK	0.546	20623360	875802	3905101
35	OK	0.531	20606976	675241	3303453
36	OK	0.531	20619264	782803	3626112
37	OK	0.578	20619264	1038992	4403247
38	OK	0.562	20627456	1038702	4402562
39	OK	0.562	20623360	1039245	4403712

Задача 2. Число инверсий

Инверсией в последовательности чисел A называется такая ситуация, когда i<j, а Ai>Aj.

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.

Пример

input.txt	output.txt
10 1 8 2 1 4 7 3 2 3 6	17

Код программы (C++)

```
#include <fstream>
using namespace std;


//"Сливает" два отсортированных по возрастанию массива в один отсортированный по возрастанию
long* merge_arrays(long* array1, int length1, long* array2, int length2, long long &inversions) {
	int result_length = length1 + length2;
	long* result_array = new long[result_length];

	int i = 0;
	int j = 0;

	for (int counter = 0; counter < result_length; counter++) {
		if ((array1[i] <= array2[j] || j == length2) && i != length1) { //Если элемент "левого" массива меньше элемента "правого" или правый закончился,
			result_array[counter] = array1[i];			//и при условии, что не закончился левый, элемент левого массива ставится на следующую позицию
			i++;							//результирующего массива
		}
		else {								//В обратном случае в результирующий массив добавляется элемент правого массива
			result_array[counter] = array2[j];
			inversions += length1 - i;			//Все "оставшиеся" элементы левого массива будут больше этого элемента правого массива
			j++;
		}
	}
	return result_array;
}

//Сортирует массив определённой длины
long* sort_array(long* array, int length, long long &inversions) {
	if (length == 1) {
		return array;
	}

	//Деление массива на два подмассива, где первый <= второго
	int left_array_size, right_array_size;
	if (length % 2 == 1) {
		left_array_size = length / 2;
		right_array_size = length / 2 + 1;
	}
	else {
		left_array_size = length / 2;
		right_array_size = length / 2;
	}
	long* left_array = new long[left_array_size];
	long* right_array = new long[right_array_size];

	for (int i = 0; i < length; i++) {
		if (i < left_array_size) {
			left_array[i] = array[i];
		}
		else {
			right_array[i - left_array_size] = array[i];
		}
	}
```

```
//Вызов функции для каждого из двух подмассивов для их сортировки
	left_array = sort_array(left_array, left_array_size, inversions);
	right_array = sort_array(right_array, right_array_size, inversions);

	//Слияние отсортированных массивов в один
	return merge_arrays(left_array, left_array_size, right_array, right_array_size,
inversions);
}

long* read_array_from_file(string file_name, int &size) {
	ifstream input(file_name);
	input >> size;
	long* array = new long[size];

	for (int i = 0; i < size; i++) {
		input >> array[i];
	}

	input.close();
	return array;
}

int main() {

	int size;

	long* array = read_array_from_file("input.txt", size);
	long long inversions = 0;

	array = sort_array(array, size, inversions);

	ofstream output("output.txt");
	output << inversions;
	output.close();
	return 0;
}
```

Бенчмарк (задача 2)

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.156	20635648	1039245	10
1	OK	0.000	2367488	25	2
2	OK	0.000	2379776	6	1
3	OK	0.000	2367488	8	1
4	OK	0.000	2363392	8	1
5	OK	0.000	2375680	42	1
6	OK	0.015	2367488	43	2
7	OK	0.015	2363392	51	1
8	OK	0.015	2375680	45	2
9	OK	0.000	2367488	105	2
10	OK	0.015	2379776	110	2
11	OK	0.000	2371584	107	2
12	OK	0.015	2392064	461	1
13	OK	0.015	2392064	560	4
14	OK	0.000	2392064	388	1
15	OK	0.000	2387968	408	4
16	OK	0.015	2400256	1042	4
17	OK	0.000	2404352	1043	4
18	OK	0.000	2392064	1044	4
19	OK	0.000	2527232	5587	1
20	OK	0.000	2535424	6733	6
21	OK	0.015	2527232	4737	1
22	OK	0.015	2527232	5685	6
23	OK	0.000	2523136	10383	6
24	OK	0.015	2527232	10421	6
25	OK	0.015	2527232	10420	6
26	OK	0.015	4055040	65880	1
27	OK	0.015	4059136	77550	8
28	OK	0.015	4042752	57488	1
29	OK	0.000	4042752	68090	8
30	OK	0.031	4042752	103872	8
31	OK	0.015	4042752	103940	8
32	OK	0.015	4055040	103842	8
33	OK	0.125	20627456	758839	1
34	OK	0.125	20631552	875802	10
35	OK	0.125	20627456	675241	1
36	OK	0.125	20635648	782803	10
37	OK	0.156	20627456	1038992	10
38	OK	0.140	20627456	1038702	10
39	OK	0.156	20635648	1039245	10

Задача 3. Анти-quick sort

Для сортировки последовательности чисел широко используется быстрая сортировка — QuickSort. Далее приведена программа, которая сортирует массив a, используя этот алгоритм.

```
var a : array [1..N] of integer;

procedure QSort(left, right : integer);
var i, j, key, buf : integer;
begin
   key := a[(left + right) div 2];
   i := left;
   j := right;
   repeat
      while a[i] < key do
         inc(i);
      while key < a[j] do
         dec(j);
      if i <= j then begin
         buf := a[i];
         a[i] := a[j];
         a[j] := buf;
         inc(i);
         dec(j);
      end;
   until i > j;
   if left < j then QSort(left, j);
   if i < right then QSort(i, right);
end;
begin
   ...
   QSort(1, N);
end.
```

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 ($1 \le n \le 10^6$).

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.

Пример

input.txt	output.txt
3	1 3 2

Код программы (C++)

```
#include <fstream>
using namespace std;

void swop(long* a, long* b) {
	long temp = *a;
	*a = *b;
	*b = temp;
}


int main() {

	ifstream input("input.txt");
	long n;
	input >> n;
	input.close();

	long* array = new long[n];
	//Каждый следующий элемент вставляется в конец массива, а затем меняется с центральным элементом
	//Тогда при сортировке будет максимальное число рекурсии - N
	for (long i = 0; i < n; i++)
	{
		array[i] = i + 1;
		if (i > 1) {
			swop(&array[i], &array[i / 2]);
		}
	}


	ofstream output("output.txt");

	for (long i = 0; i < n; i++) {
		output << array[i];
		if (i != n - 1) output << " ";
	}

	output.close();

	return 0;

}
```

Бенчмарк (задача 3)

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.750	6381568	9	6888895
1	OK	0.000	2588672	3	5
2	OK	0.015	2592768	3	1
3	OK	0.000	2588672	3	3
4	OK	0.031	2588672	3	7
5	OK	0.000	2605056	3	9
6	OK	0.000	2592768	3	11
7	OK	0.031	2592768	3	13
8	OK	0.015	2588672	3	15
9	OK	0.000	2588672	3	17
10	OK	0.000	2363392	4	20
11	OK	0.000	2383872	4	35
12	OK	0.000	2367488	5	291
13	OK	0.000	2375680	6	3892
14	OK	0.015	2408448	7	48899
15	OK	0.015	2404352	7	48893
16	OK	0.093	2867200	8	756194
17	OK	0.171	3338240	8	1556238
18	OK	0.359	4239360	8	3151811
19	OK	0.750	6381568	8	6888887
20	OK	0.750	6381568	9	6888895

Задача 4. К-ая порядковая статистика

Дан массив из n элементов. Какие числа являются k1-ым, (k1+1)-ым, …, k2-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 k1 и k2, при этом $2 \le n \le 4 \cdot 10^7$, $1 \le k1 \le k2 \le n$, $k2 - k1 < 200$.

Во второй строке находятся числа A, B, C, a1, a2, по модулю не превосходящие 109. 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.

Обращаем Ваше внимание, что массив из $4 \cdot 10^7$ 32-битных целых чисел занимает в памяти **160 мегабайт**! Будьте аккуратны!

Подсказка: эту задачу лучше всего решать модификацией быстрой сортировки. Однако сортировка массива целиком по времени, скорее всего, не пройдет, поэтому нужно подумать, как модифицировать быструю сортировку, чтобы не сортировать те части массива, которые не нужно сортировать.

Формат выходного файла

В первой и единственной строке выходного файла выведите k1-ое, (k1+1)-ое, …, k2-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

Код программы (C++)

```
#include <fstream>
using namespace std;

void swop(long* a, long* b) {
	long temp = *a;
	*a = *b;
	*b = temp;
}
//Алгоритм быстрой сортировки
void quicksort(long* array, long left, long right, long k1, long k2) {
	//Если нужная область полностью находится в левом или правом подмассиве, то
сортировка второго уже не нужна
	if (left > k2 || right < k1) return;
	long key = array[(left + right) / 2];
	long i = left;
	long j = right;
	do {
		while (array[i] < key) {
			i++;
		}
		while (array[j] > key) {
			j--;
		}
		if (i <= j) {
			swop(&array[i], &array[j]);
			i++;
			j--;
		}
	} while (i < j);
	if (left < j) quicksort(array, left, j, k1, k2);
	if (i < right) quicksort(array, i, right, k1, k2);
}

int main() {
	long n, k1, k2, A, B, C, a1, a2;

	ifstream input("input.txt");
	input >> n >> k1 >> k2 >> A >> B >> C >> a1 >> a2;
	input.close();

	long *array = new long[n];

	array[0] = a1;
	array[1] = a2;
	k1--;
	k2--;
	//Заполнение массива по формуле из задания
	for (long i = 2; i < n; i++) {
		array[i] = A * array[i - 2] + B * array[i - 1] + C;
	}
	//Сортировка массива
	quicksort(array, 0, n - 1, k1, k2);

	ofstream output("output.txt");
	//Вывод элементов от k1 до k2
	for (long i = k1; i <= k2; i++) {
		output << array[i];
		if (i != k2) output << " ";
	}
	output.close();
	return 0;
}
```

Бенчмарк (задача 4)

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.687	162385920	54	2399
1	OK	0.000	2367488	18	5
2	OK	0.000	2367488	28	8
3	OK	0.015	2379776	32	3
4	OK	0.000	2363392	33	4
5	OK	0.000	2367488	32	9
6	OK	0.000	2363392	33	4
7	OK	0.000	2367488	32	18
8	OK	0.015	2367488	32	20
9	OK	0.000	2367488	25	299
10	OK	0.000	2367488	22	381
11	OK	0.000	2379776	23	476
12	OK	0.000	2367488	35	11
13	OK	0.000	2363392	38	10
14	OK	0.000	2367488	36	1073
15	OK	0.015	2367488	36	560
16	OK	0.000	2367488	37	219
17	OK	0.000	2408448	24	399
18	OK	0.015	2408448	28	1199
19	OK	0.000	2408448	29	1399
20	OK	0.031	2420736	37	11
21	OK	0.000	2408448	45	10
22	OK	0.000	2408448	38	2399
23	OK	0.000	2404352	39	2399
24	OK	0.000	2408448	44	2199
25	OK	0.000	2408448	43	2199
26	OK	0.000	2416640	41	675
27	OK	0.015	2764800	28	599
28	OK	0.000	2772992	31	1399
29	OK	0.000	2768896	32	1599
30	OK	0.000	2768896	37	11
31	OK	0.000	2785280	48	10
32	OK	0.000	2764800	40	2399
33	OK	0.000	2768896	40	2399
34	OK	0.015	2764800	47	2199
35	OK	0.015	2768896	46	2199
36	OK	0.000	2781184	45	199
37	OK	0.015	6381568	32	799
38	OK	0.000	6381568	34	1599
39	OK	0.015	6381568	35	1799
40	OK	0.015	6385664	38	11
41	OK	0.015	6377472	49	10
42	OK	0.015	6381568	40	2399
43	OK	0.015	6381568	40	2002
44	OK	0.015	6377472	49	2199
45	OK	0.000	6377472	47	2199
46	OK	0.015	6381568	48	559
47	OK	0.312	162381824	33	799
48	OK	0.296	162377728	39	1999
49	OK	0.359	162369536	40	2199

50	OK	0.468	162381824	40	11
51	OK	0.312	162385920	52	10
52	OK	0.390	162381824	42	2399
53	OK	0.484	162381824	42	2399
54	OK	0.421	162377728	54	2199
55	OK	0.515	162381824	54	2199
56	OK	0.687	162381824	52	1075
57	OK	0.453	162381824	53	2199
58	OK	0.531	162381824	52	2075
59	OK	0.390	162381824	54	2034
60	OK	0.359	162381824	53	1858
61	OK	0.578	162381824	51	2207
62	OK	0.312	162385920	49	2188
63	OK	0.390	162385920	53	2056
64	OK	0.531	162381824	54	1990
65	OK	0.500	162381824	50	2003
66	OK	0.515	162381824	52	1792
67	OK	0.312	162385920	54	1929

Задача 5. Сортировка пугалом

«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.

Примеры

input.txt	output.txt
3 2 2 1 3	NO
5 3 1 5 3 4 1	YES

Код программы (C++)

```
#include <fstream>
using namespace std;

void swop(long* a, long* b) {
	long temp = *a;
	*a = *b;
	*b = temp;
}

//Быстрая сортировка, модернизированная таким образом, чтобы сортировать массив,
содержащий каждый K-тый элемент исходного массива
void quicksort(long* array, long left, long right, int step) {
	if (right == left) return;
	int middle;
	if (step == 1) {
		middle = (left + right) / 2;
	}
	else {
		int offset = left % step;
		middle = ((left / step + right / step) / 2) * step + offset;
	}
	long key = array[middle];
	long i = left;
	long j = right;
	do {
		while (array[i] < key) {
			i += step;
		}
		while (array[j] > key) {
			j -= step;
		}
		if (i <= j) {
			swop(&array[i], &array[j]);
			i += step;
			j -= step;
		}
	} while (i < j);
	if (left < j) quicksort(array, left, j, step);
	if (i < right) quicksort(array, i, right, step);
}

//Разбивает массив на K частей и вызывает сортировку для каждой
void sort(long *array, int size, int k) {
	if (k == 1) {
		quicksort(array, 0, size - 1, k);
		return;
	}
	int j;
	for (int i = 0; i < k; i++) {
		j = i;
		while (j < size) {
			j += k;
		}
		j -= k;
		quicksort(array, i, j, k);
	}
}
```

```cpp
//Проверяет, отсортирован ли массив по неубыванию
bool is_sorted(long *array, int size) {
	for (int i = 1; i < size; i++) {
		if (array[i] < array[i - 1]) return false;
	}
	return true;
}

int main() {
	int n, k;

	ifstream input("input.txt");
	input >> n >> k;

	long *array = new long[n];
	for (int i = 0; i < n; i++) {
		input >> array[i];
	}

	input.close();

	sort(array, n, k);


	ofstream output("output.txt");
	if (is_sorted(array, n)) {
		output << "YES";
	}
	else {
		output << "NO";
	}
	output.close();

	return 0;
}
```

Бенчмарк (задача 5)

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.140	2748416	1039313	3
1	OK	0.000	2330624	12	2
2	OK	0.000	2347008	16	3
3	OK	0.000	2342912	112	3
4	OK	0.000	2326528	111	2
5	OK	0.000	2330624	112	3
6	OK	0.000	2330624	112	2
7	OK	0.031	2342912	109	3
8	OK	0.015	2330624	112	2
9	OK	0.000	2330624	110	3
10	OK	0.000	2330624	111	2
11	OK	0.015	2330624	108	3
12	OK	0.000	2338816	11674	3
13	OK	0.015	2334720	11707	2
14	OK	0.000	2338816	11712	3
15	OK	0.031	2351104	11754	2
16	OK	0.000	2342912	11708	3
17	OK	0.031	2338816	11740	2
18	OK	0.000	2338816	11726	3
19	OK	0.000	2338816	11680	2
20	OK	0.000	2338816	11741	3
21	OK	0.015	2383872	128736	3
22	OK	0.015	2383872	128832	2
23	OK	0.015	2392064	128751	3
24	OK	0.015	2383872	128866	2
25	OK	0.015	2379776	128700	3
26	OK	0.015	2383872	128707	2
27	OK	0.015	2383872	128729	3
28	OK	0.015	2400256	128807	2
29	OK	0.015	2379776	128784	3
30	OK	0.109	2740224	1039313	3
31	OK	0.125	2744320	1038610	2
32	OK	0.140	2744320	1038875	3
33	OK	0.109	2748416	1038723	2
34	OK	0.109	2748416	1038749	3
35	OK	0.125	2744320	1038747	2
36	OK	0.125	2744320	1039043	3
37	OK	0.109	2732032	1039210	2
38	OK	0.109	2744320	1038967	3