Лабораторная работа № 2

по дисциплине «Алгоритмы и структуры данных»

Openedu – неделя 2

Подготовил:

студент группы P3217

Бураков Илья Алексеевич

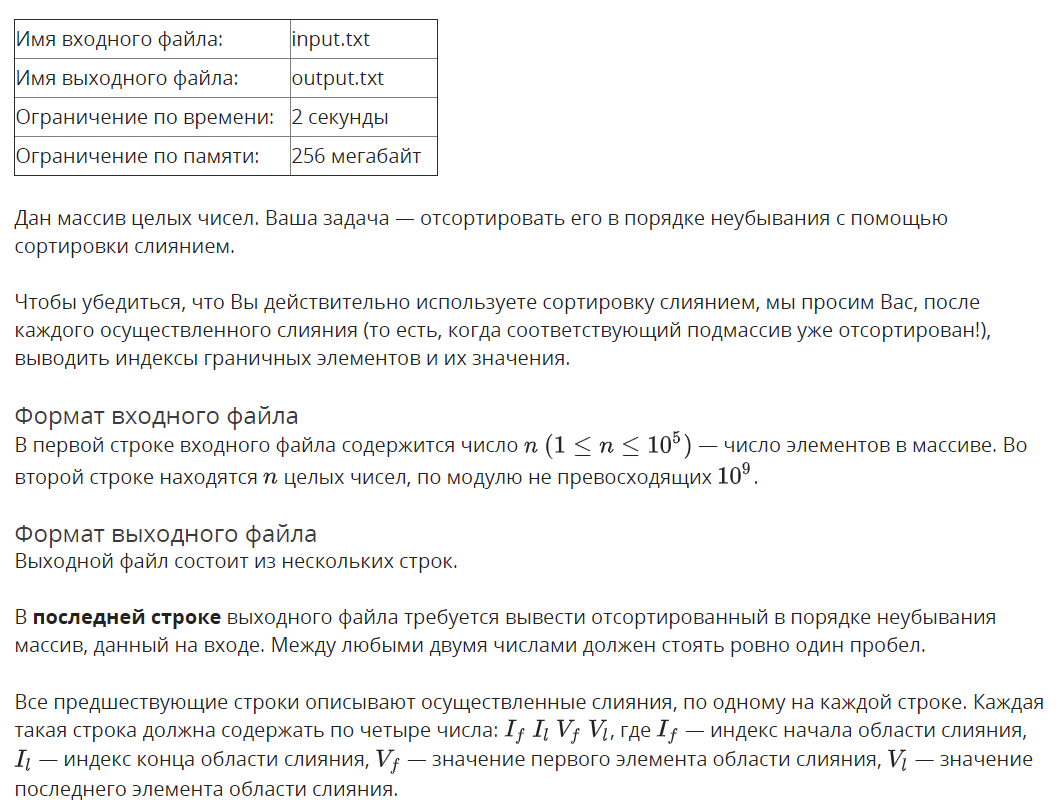
Преподаватели:

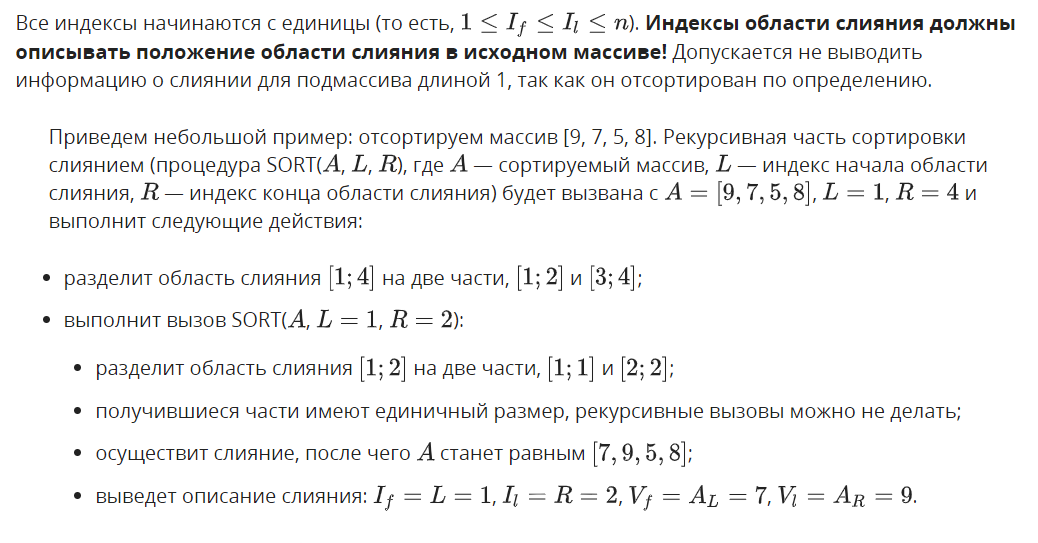
Романов Алексей Андреевич

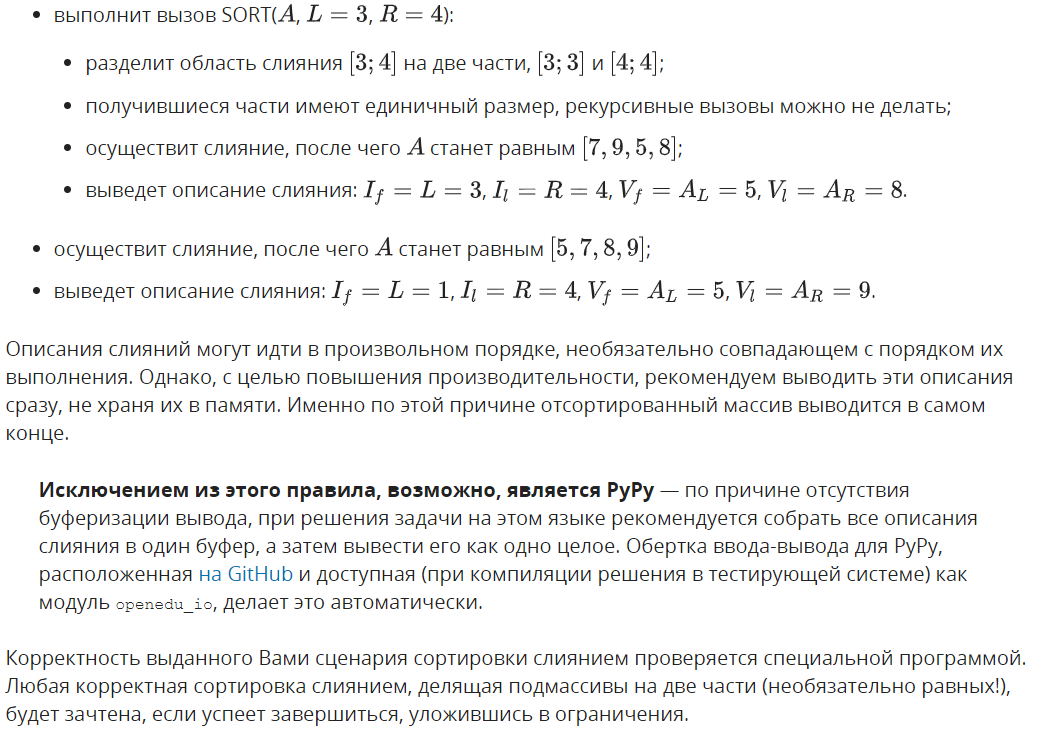
Волчек Дмитрий Геннадьевич

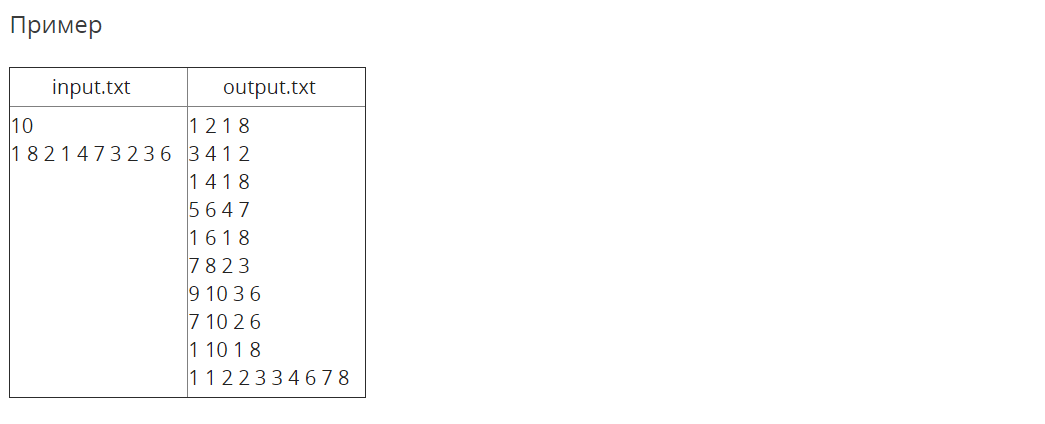
## Сортировка слиянием

## Условие









## Решение

openedu/week2/lab2\_1.cpp

#include "edx-io.hpp"

#include <vector>

using namespace std;

void merge\_sort(vector<int> &arr, int left, int right) {

// left subarray index in inclusive

// right subarray index is exclusive (!)

// length of the subarray to sort

int subarray\_n = right - left;

if (subarray\_n == 1) {

// already sorted, job done

return;

}

// beginning of left and right subarrays

int left\_beginning = left;

int right\_beginning = left + subarray\_n / 2;

// recursively sort subarrays

merge\_sort(arr, left\_beginning, right\_beginning);

merge\_sort(arr, right\_beginning, right);

// merge sorted subarrays

auto merged\_arr = vector<int>(subarray\_n);

int i = left\_beginning;

int j = right\_beginning;

for (int k = 0; k < subarray\_n;) {

if (j == right || (i != right\_beginning && arr[i] < arr[j])) {

// left subarray wins for current k

merged\_arr[k++] = arr[i++];

} else {

// right subarray wins for current k

merged\_arr[k++] = arr[j++];

}

}

// place merged\_arr

copy\_n(merged\_arr.begin(), subarray\_n, arr.begin() + left);

// print required info

// (right - 1) -- inclusive

// (right - 1) + 1 -- indexing from 1

io << left + 1 << " " << right << " " << merged\_arr.front() << " " << merged\_arr.back() << "\n";

}

int main() {

// read N

int n;

io >> n;

// read input array

auto arr = vector<int>(n);

for (auto &e : arr) {

io >> e;

}

// sorting

merge\_sort(arr, 0, n);

// printing sorted array

for (auto e : arr) {

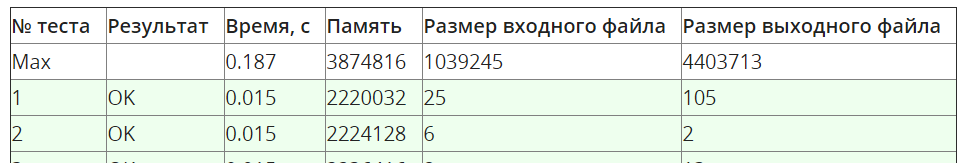
io << e << " ";

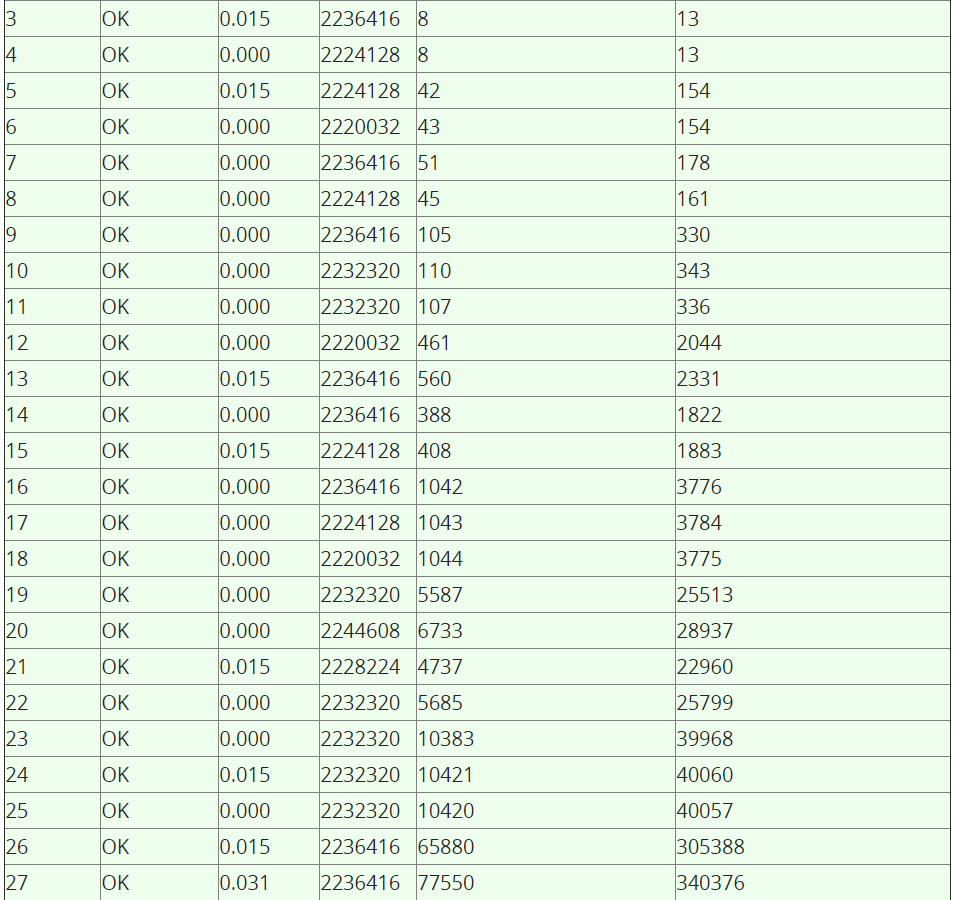
}

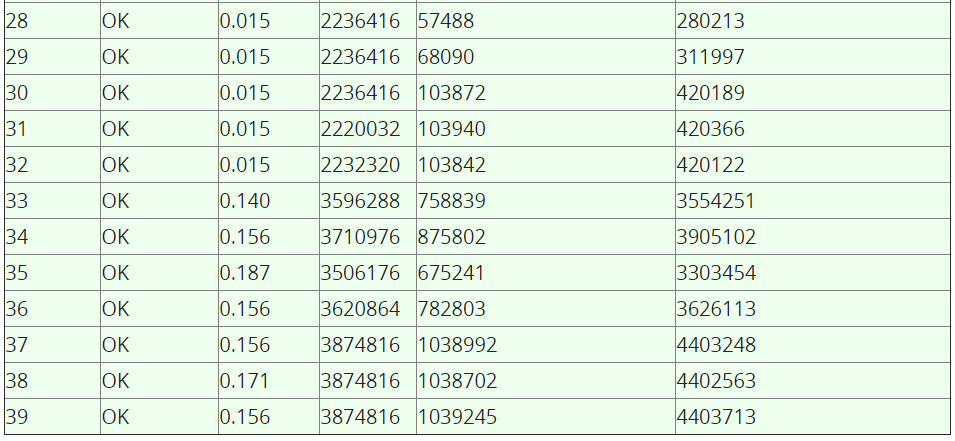
return 0;

}

## Результаты

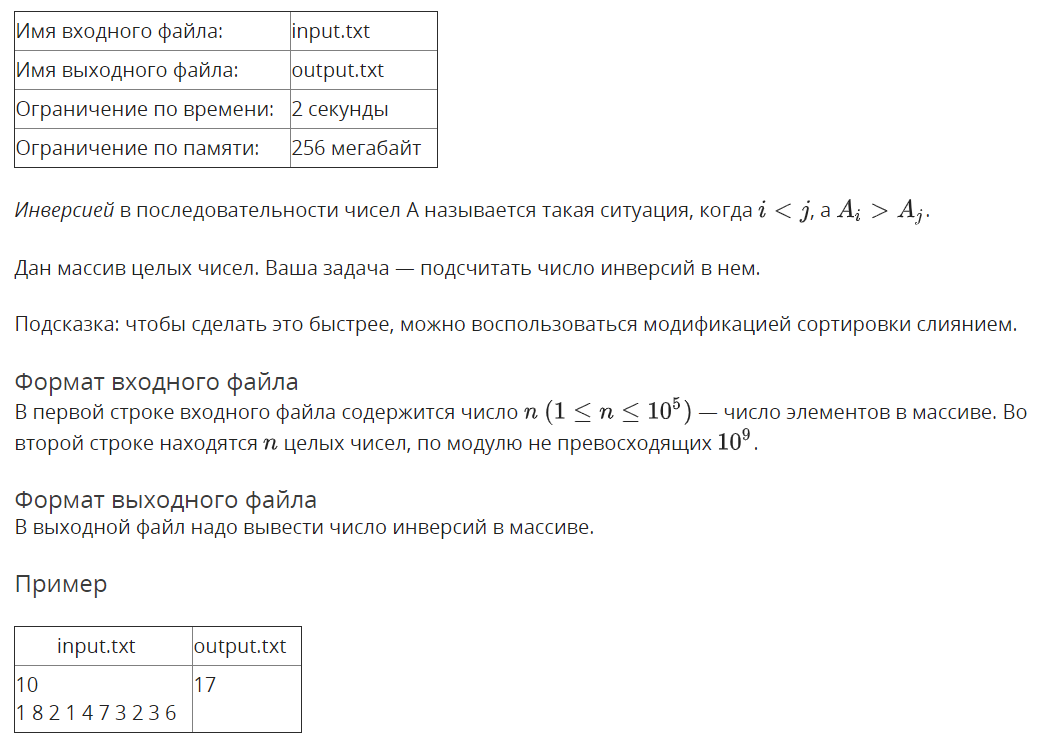






## Число инверсий

## Условие



## Решение

openedu/week2/lab2\_2.cpp

#include "edx-io.hpp"

#include <vector>

using namespace std;

long long count\_inversions(vector<int> &arr, int left, int right) {

// left subarray index in inclusive

// right subarray index is exclusive (!)

// length of the subarray to sort

int subarray\_n = right - left;

if (subarray\_n == 1) {

// already sorted, job done

return 0;

}

// beginning of left and right subarrays

int left\_beginning = left;

int right\_beginning = left + subarray\_n / 2;

// recursively sort subarrays

long long result = 0;

result += count\_inversions(arr, left\_beginning, right\_beginning);

result += count\_inversions(arr, right\_beginning, right);

// merge sorted subarrays

auto merged\_arr = vector<int>(subarray\_n);

int i = left\_beginning;

int j = right\_beginning;

for (int k = 0; k < subarray\_n;) {

if (j == right || (i != right\_beginning && arr[i] <= arr[j])) {

// left subarray wins for current k

merged\_arr[k++] = arr[i++];

} else {

// right subarray wins for current k

merged\_arr[k++] = arr[j++];

// Since right subarray won, all left subarray elements larger than the element from

// right subarray will form an inversion. Add their quantity to result.

result += right\_beginning - i;

}

}

// place merged\_arr

copy\_n(merged\_arr.begin(), subarray\_n, arr.begin() + left);

return result;

}

int main() {

// read N

int n;

io >> n;

// read input array

auto arr = vector<int>(n);

for (auto &e : arr) {

io >> e;

}

// sorting

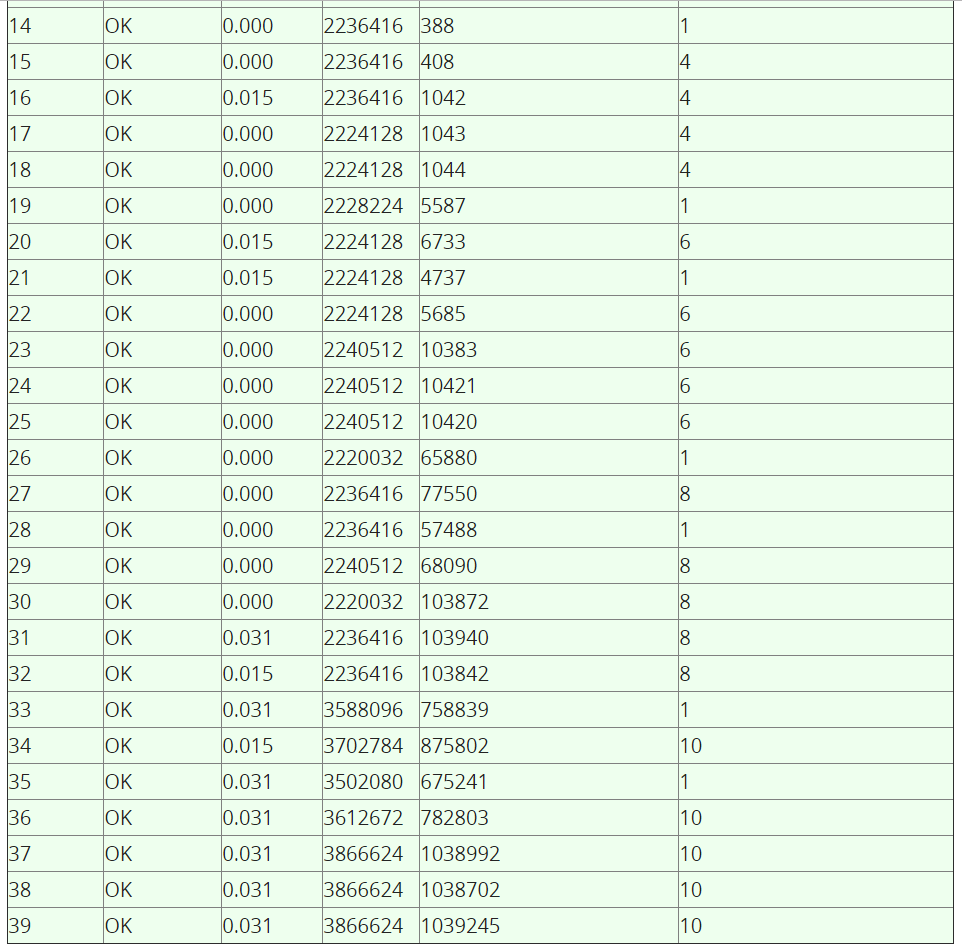
io << count\_inversions(arr, 0, n);

return 0;

}

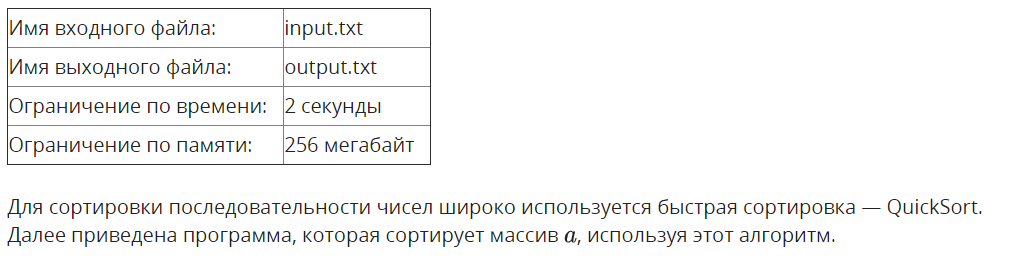
## Результаты





## Анти-quick sort

## Условие



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);

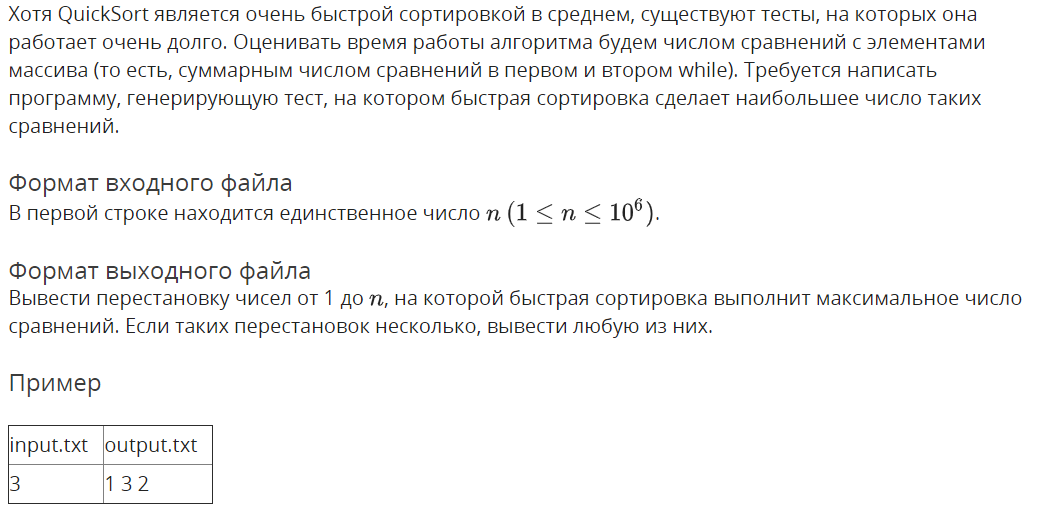
end;

begin

...

QSort(1, N);

end.



## Решение

openedu/week2/lab2\_3.cpp

#include "edx-io.hpp"

#include <vector>

using namespace std;

void swap(vector<int> &arr, int a, int b) {

int t = arr[a];

arr[a] = arr[b];

arr[b] = t;

}

int main() {

// read N

int n;

io >> n;

// read input array

auto arr = vector<int>(n);

for (int i = 0; i < n; i++) {

arr[i] = i + 1;

}

// generate array

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

swap(arr, i, i / 2);

}

// print arr

//reverse(arr.begin(), arr.end());

for (auto e : arr) {

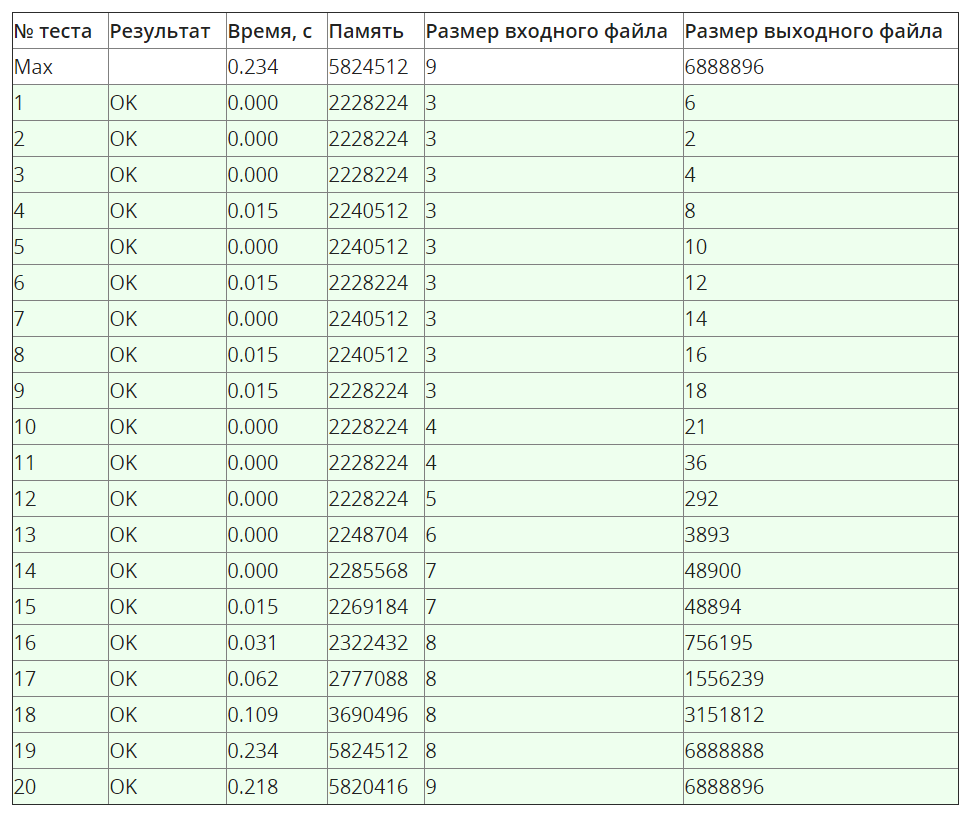
io << e << " ";

}

return 0;

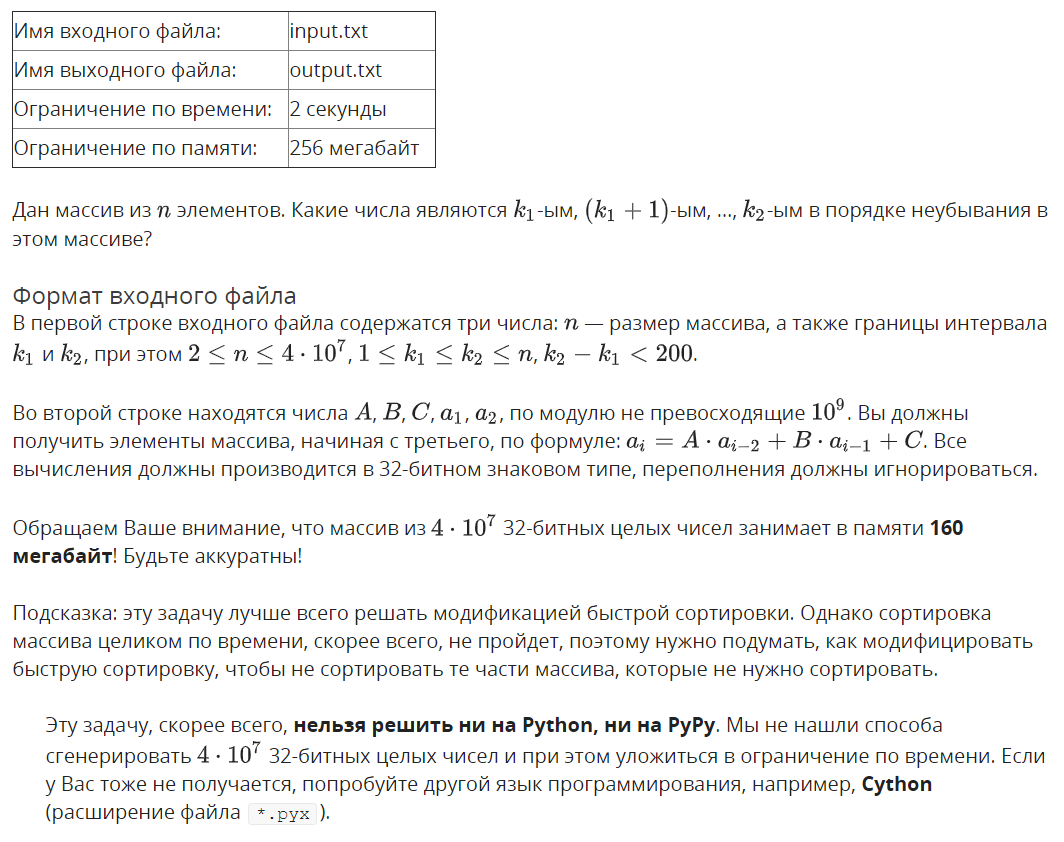
}

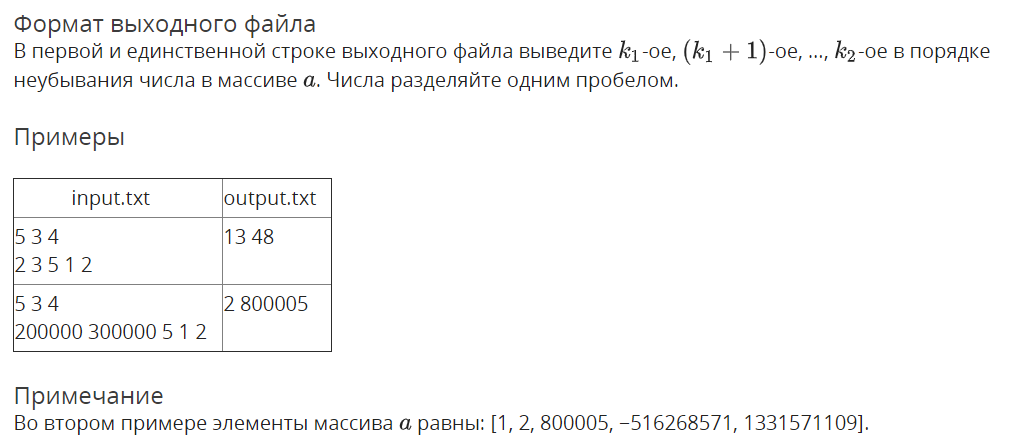
## Результаты



## K-ая порядковая статистика

## Условие





## Решение

openedu/week2/lab2\_4.cpp

#include "edx-io.hpp"

#include <vector>

using namespace std;

int k1, k2;

int hoare\_patrition(vector<int> &arr, int left, int right) {

int pivot = arr[(left + right) / 2];

int i = left - 1;

int j = right + 1;

while (true) {

do i++; while (arr[i] < pivot);

do j--; while (arr[j] > pivot);

if (i >= j) {

return j;

}

int t = arr[i];

arr[i] = arr[j];

arr[j] = t;

}

}

void quick\_sort(vector<int> &arr, int left, int right) {

// if we don't need to sort that subarray, skip it!

if (right < k1 || left > k2) return;

if (left < right) {

int middle = hoare\_patrition(arr, left, right);

quick\_sort(arr, left, middle);

quick\_sort(arr, middle + 1, right);

}

}

int main() {

// read input

int n, A, B, C;

io >> n >> k1 >> k2;

k1--;

k2--;

auto arr = vector<int>(n);

io>> A >> B >> C >> arr[0] >> arr[1];

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

arr[i] = A \* arr[i - 2] + B \* arr[i - 1] + C;

}

// sort array

quick\_sort(arr, 0, n - 1);

// print needed part

for (int i = k1; i <= k2; i++) {

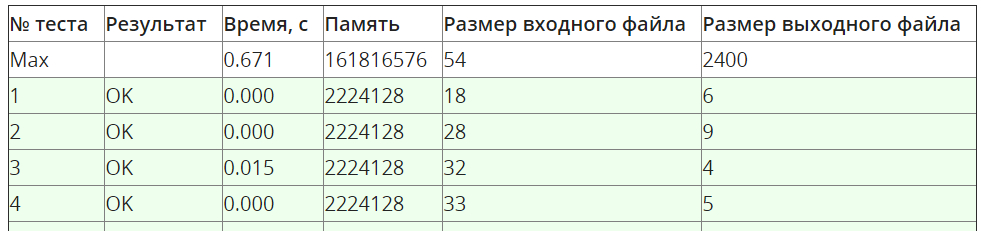
io << arr[i]<< " ";

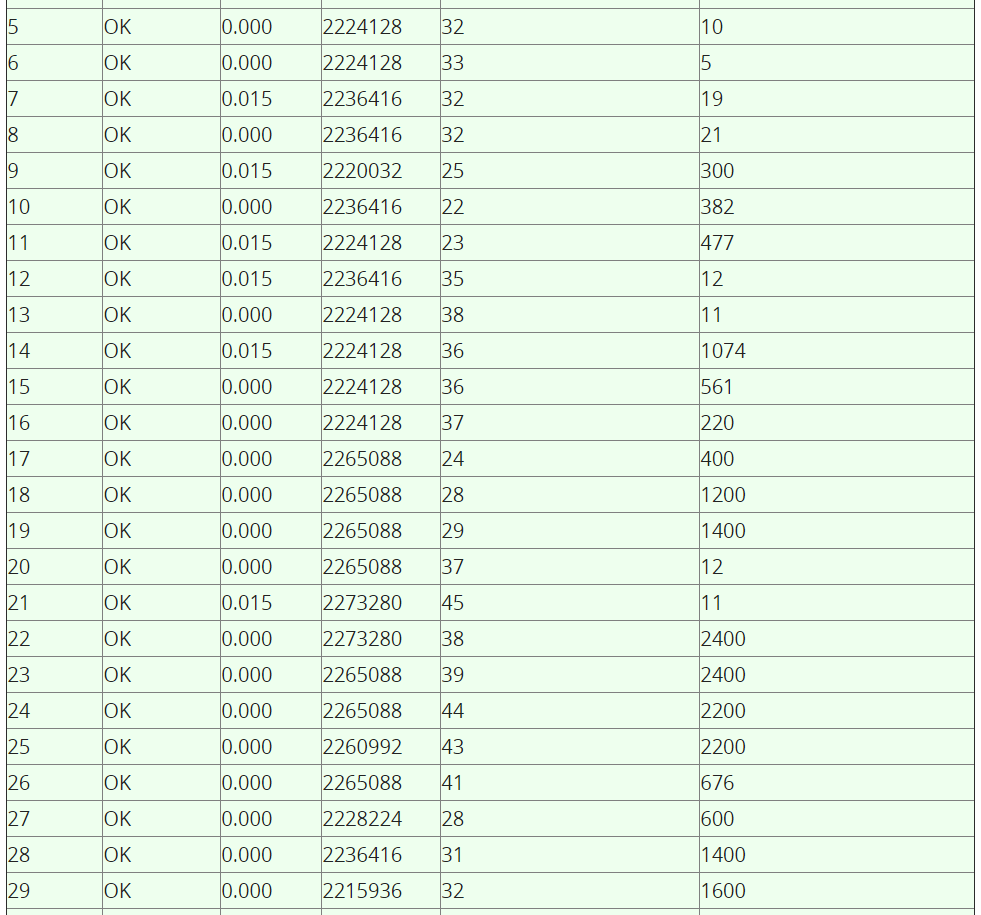
}

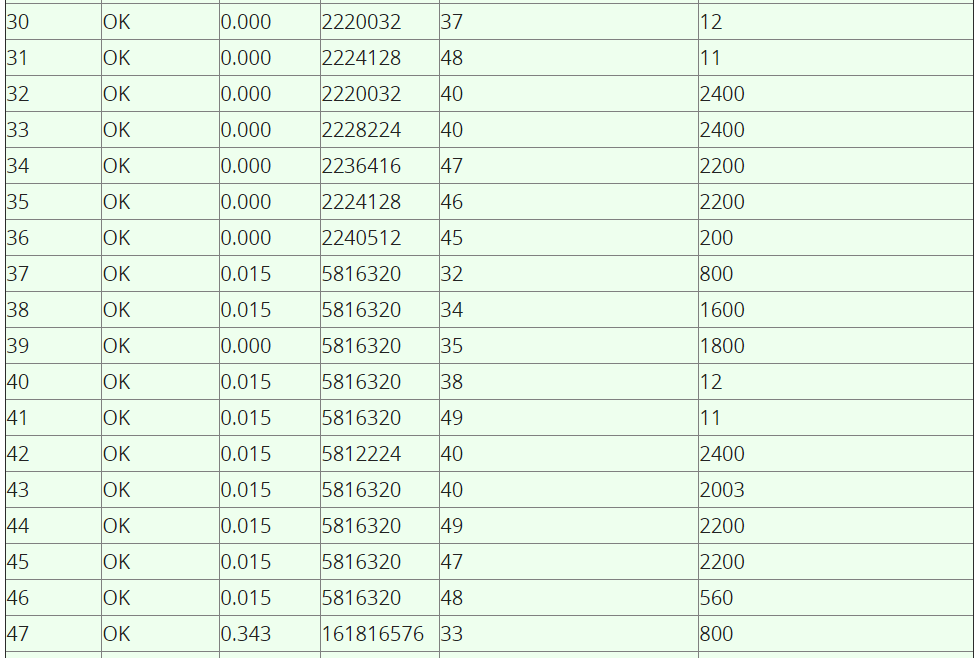
return 0;

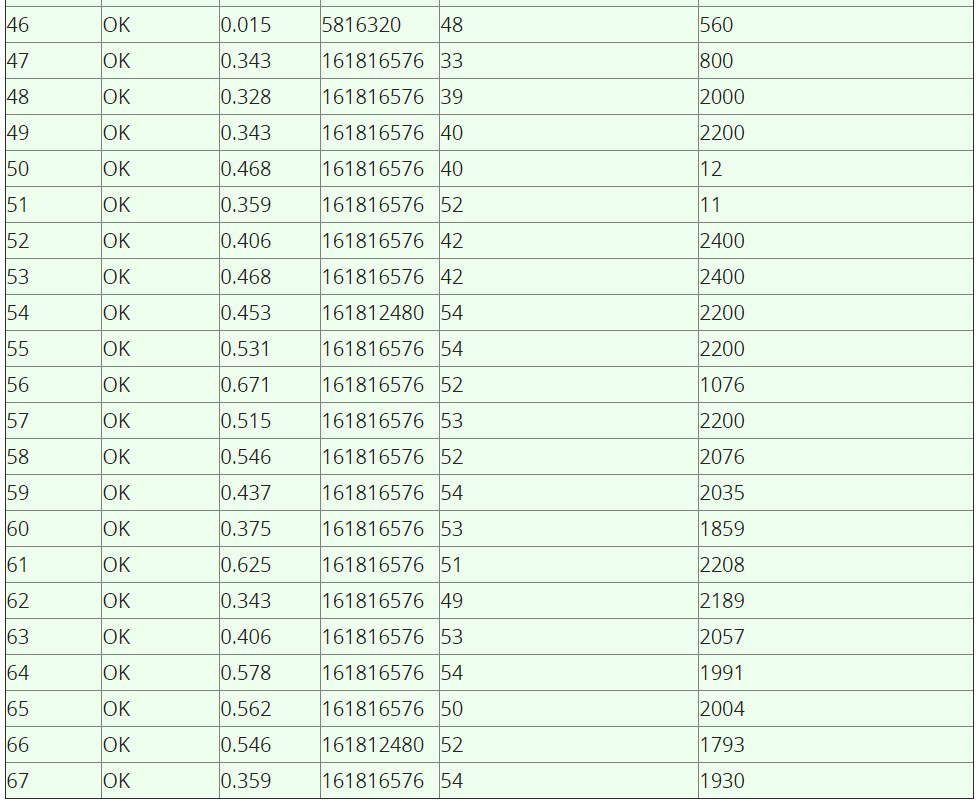
}

## Результаты



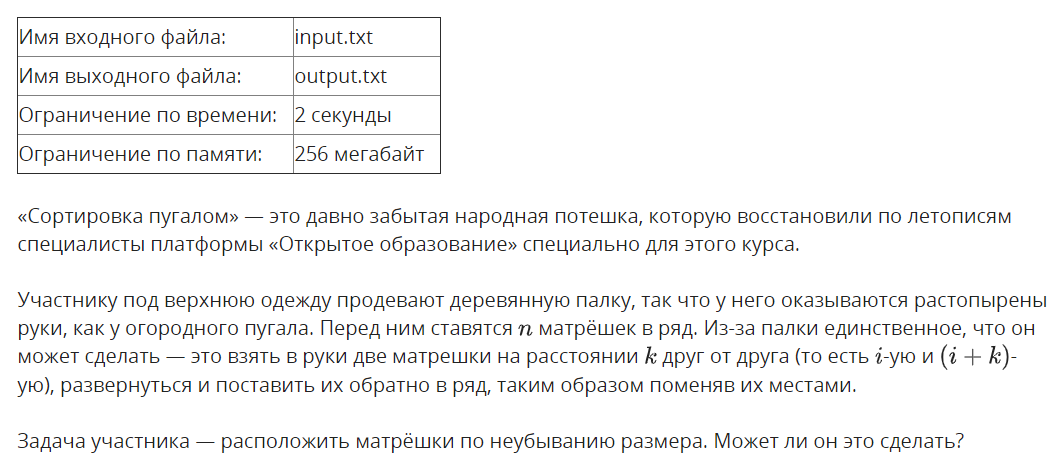


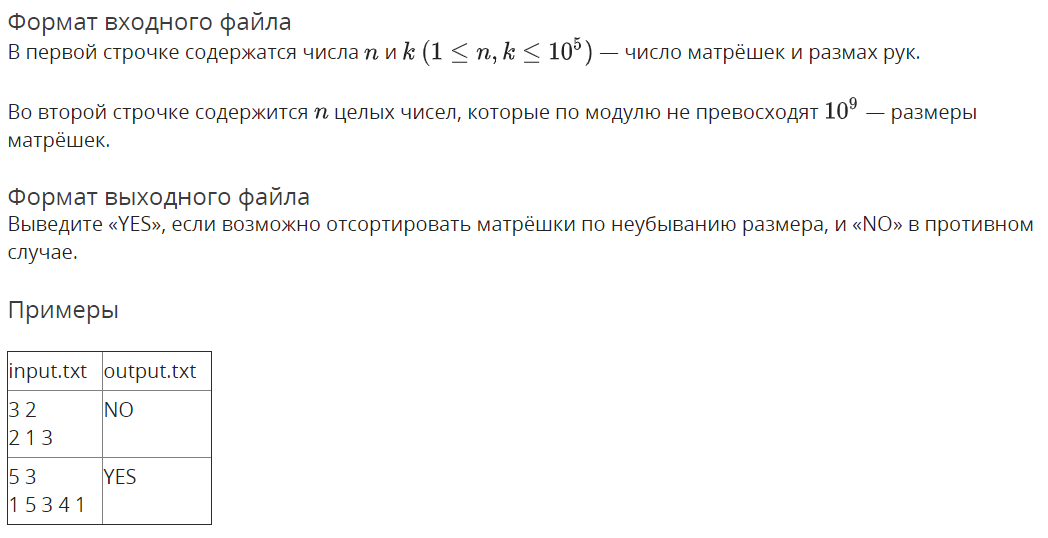




## Сортировка пугалом

## Условие





## Решение

openedu/week2/lab2\_5.cpp

#include "edx-io.hpp"

#include <vector>

#include <algorithm>

using namespace std;

int n, k;

void sort\_k\_slice(vector<long long> &arr, int start) {

auto slice = vector<long long>();

slice.reserve(n / k);

// extract a slice

for (int i = start; i < n; i += k) {

slice.push\_back(arr[i]);

}

sort(slice.begin(), slice.end());

// put it back

for (int i = 0; i < slice.size(); i++) {

arr[start + i \* k] = slice[i];

}

}

int main() {

// read input

io >> n >> k;

// EDGE CASES

// if step is 1, ANY array can be sorted

// if n is 1, array is sorted by definition

if (k == 1 || n == 1) {

io << "YES";

return 0;

}

// initializing array with input

auto arr = vector<long long>(n);

for (int i = 0; i < n; i++) {

io >> arr[i];

}

// SEPARATELY sort slices of corresponding k (in pythonic terms - [i::k])

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

sort\_k\_slice(arr, i);

}

// check if array is sorted.

// if it's not, then it's impossible.

bool is\_pugalo\_crying = false;

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

if (arr[i] < arr[i - 1]) {

is\_pugalo\_crying = true;

break;

}

}

// print needed part

io << (is\_pugalo\_crying ? "NO" : "YES");

return 0;

}

## Результаты

