**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 «Aлгоритмы и структуры данных»

**ОТЧЁТ**

по лабораторной работе Разделяй и властвуй (Stepic)

Студенка Кузенкова Елизавета группы P3217

Преподаватель Муромцев Дмитрий Ильич

Санкт-Петербург

2019 г.

Содержание

[Задача 1: двоичный поиск 3](#_Toc8427946)

[Исходный код к задаче 1 3](#_Toc8427947)

[Задача 2: число инверсий 4](#_Toc8427948)

[Исходный код к задаче 2 4](#_Toc8427949)

[Задача 3: точки и отрезки 5](#_Toc8427950)

[Исходный код к задаче 3 5](#_Toc8427951)

[Задача 4: сортировка подсчетом 7](#_Toc8427952)

[Исходный код к задаче 4 7](#_Toc8427953)

# Задача 1: двоичный поиск

В первой строке даны целое число 1≤*n*≤105 и массив *A*[1…*n*] из *n* различных натуральных чисел, не превышающих 109, в порядке возрастания, во второй — целое число 1≤*k*≤105 и *k* натуральных чисел *b*1,…,*bk*, не превышающих 109. Для каждого *i* от 1 до *k* необходимо вывести индекс 1≤*j*≤*n*, для которого *A*[*j*]=*bi*, или −1, если такого *j* нет.

**Sample Input:**

5 1 5 8 12 13

5 8 1 23 1 11

**Sample Output:**

3 1 -1 1 -1

# Исходный код к задаче 1

#include <iostream>

#include <vector>

int my\_bsearch(int val, std::vector<int>& mas) {

int l = 0;

int r = mas.size() - 1;

if (val > mas.at(r)) return -1;

while (l <= r) {

int m = l + (r - l) / 2;

unsigned int elem = mas.at(m);

if (elem == val) return m + 1; // Для нумерации с 1

if (val > elem) l = m + 1;

else r = m - 1;

}

return -1;

}

int main()

{

int num\_count = 0;

std::cin >> num\_count;

std::vector<int> mas;

while (num\_count-- > 0) {

int val = 0;

std::cin >> val;

mas.push\_back(val);

}

std::cin >> num\_count;

std::vector<int> nums\_to\_seach;

while (num\_count-- > 0) {

int val = 0;

std::cin >> val;

nums\_to\_seach.push\_back(val);

}

for (auto val : nums\_to\_seach) {

std::cout << my\_bsearch(val, mas) << " ";

}

std::cout << std::endl;

std::cout.flush();

return 0;

}

# Задача 2: число инверсий

Первая строка содержит число 1≤*n*≤105, вторая — массив *A*[1…*n*], содержащий натуральные числа, не превосходящие 109. Необходимо посчитать число пар индексов 1≤*i*<*j*≤*n*, для которых *A*[*i*]>*A*[*j*]. (Такая пара элементов называется инверсией массива. Количество инверсий в массиве является в некотором смысле его мерой неупорядоченности: например, в упорядоченном по неубыванию массиве инверсий нет вообще, а в массиве, упорядоченном по убыванию, инверсию образуют каждые два элемента.)

**Sample Input:**

5

2 3 9 2 9

**Sample Output:**

2

# Исходный код к задаче 2

#include <iostream>

#include <vector>

std::vector<unsigned int> merge(std::vector<unsigned int> a1, std::vector<unsigned int> a2, long& count) {

std::vector<unsigned int> result;

auto p1 = a1.begin();

auto p2 = a2.begin();

bool isP2 = false, isP1 = false;

while (true) {

if (\*p1 <= \*p2) {

result.push\_back(\*p1);

p1++;

if (p1 == a1.end()) {

isP1 = true;

break;

}

}

else {

size\_t c = (a1.end() - p1

count += c;

result.push\_back(\*p2);

p2++;

if (p2 == a2.end()) {

isP2 = true;

break;

}

}

}

if (isP1) result.insert(result.end(), p2, a2.end());

else result.insert(result.end(), p1, a1.end());

return result;

}

std::vector<unsigned int> merge\_sort(std::vector<unsigned int>& a, size\_t l, size\_t r, long& count) {

if (r == l) {

std::vector<unsigned int> vec;

vec.push\_back(a[r]);

return vec; // Если границы пересеклись

}

size\_t m = l + (r - l) / 2; // Делим пополам

return merge(merge\_sort(a, l, m, count), merge\_sort(a, m + 1, r, count), count);

}

int main()

{

unsigned int size = 0;

std::vector<unsigned int> mas;

std::cin >> size;

while (size-- >= 1) {

unsigned int val = 0;

std::cin >> val;

mas.push\_back(val);

}

long count = 0;

auto a = merge\_sort(mas, 0, mas.size() - 1, count);

std::cout << count << std::endl;

return 0;

}

# Задача 3: точки и отрезки

В первой строке задано два целых числа 1≤*n*≤50000 и 1≤*m*≤50000 — количество отрезков и точек на прямой, соответственно. Следующие *n* строк содержат по два целых числа *ai* и *bi* (*ai*≤*bi*) — координаты концов отрезков. Последняя строка содержит *m* целых чисел — координаты точек. Все координаты не превышают 108 по модулю. Точка считается принадлежащей отрезку, если она находится внутри него или на границе. Для каждой точки в порядке появления во вводе выведите, скольким отрезкам она принадлежит.

**Sample Input:**

2 3

0 5

7 10

1 6 11

**Sample Output:**

1 0 0

# Исходный код к задаче 3

#include <iostream>

#include <vector>

#include <algorithm>

#include <string>

#include <sstream>

int main()

{

// Количество отрезков и точек

unsigned int num\_lines = 0, num\_points = 0;

// масиив отрезков

std::vector<std::pair<int, int>> lines;

// массив точек

std::vector<std::pair<int, int>> points;

std::cin >> num\_lines >> num\_points;

while (num\_lines-- >= 1) {

int a = 0, b = 0;

std::cin >> a >> b;

lines.push\_back(std::make\_pair(a, b));

}

int counter = 0;

while (counter < num\_points) {

int pt = 0;

std::cin >> pt;

points.push\_back(std::make\_pair(counter, pt));

++counter;

}

// Сортируем массив точек

std::stable\_sort(points.begin(), points.end(), [](const std::pair<int, int> &e1, const std::pair<int, int> &e2) {return e1.second < e2.second;});

// Сортируем массив отрезков по левому краю

std::stable\_sort(lines.begin(), lines.end(), [](const std::pair<int, int> &e1, const std::pair<int, int> &e2) {return e1.first < e2.first;});

// Создаем новый массив из отрезков и упорядочиваем его по правому краю

std::vector<std::pair<int, int>> part;

part.insert(part.begin(), lines.begin(), lines.end());

std::stable\_sort(part.begin(), part.end(), [](const std::pair<int, int> &e1, const std::pair<int, int> &e2) {return e1.second < e2.second;});

// Массив числа отрезков содержащих каждую точку

std::vector<std::pair<int, int>> numbers;

num\_points = points.size();

// Выбираем первую точку

auto i\_pt = points.begin();

// Оптимизируем для одинаковыхх точек ????

int prev\_pt = (\*i\_pt).second;

int prev\_num = 0;

bool is\_prev = false;

auto prev\_iter\_left = lines.begin();

auto prev\_iter\_right = part.begin();

while (num\_points > numbers.size()) { // Пока пройдены не все точки

if (is\_prev && (\*i\_pt).second == prev\_pt) {

numbers.push\_back(std::make\_pair((\*i\_pt).first, prev\_num));

i\_pt++;

continue;

}

// Берем первую точку и находим позицию первого отрезка левый край которого более точки

auto pos\_left\_line = std::find\_first\_of(prev\_iter\_left, lines.end(), i\_pt, i\_pt + 1, [](const std::pair<int, int> &e1, const std::pair<int, int> &e2) {return e1.first > e2.second;});

if (pos\_left\_line == lines.begin()) { // Начала всех отрезков правее точки - пересечений нет

numbers.push\_back(std::make\_pair((\*i\_pt).first, 0));

is\_prev = true;

prev\_pt = (\*i\_pt).second;

prev\_num = 0;

i\_pt++;

continue;

}

prev\_iter\_left = lines.begin() == pos\_left\_line ? pos\_left\_line : pos\_left\_line - 1;

// Находим позицию первого отрезка правый край которого более либо равен точки

auto pos\_right\_line = std::find\_first\_of(prev\_iter\_right, part.end(), i\_pt, i\_pt + 1, [](const std::pair<int, int> &e1, const std::pair<int, int> &e2) {return e1.second >= e2.second;});

prev\_iter\_right = part.begin() == pos\_right\_line ? pos\_right\_line : pos\_right\_line - 1;

// Сохраняем найденное число отрезков

numbers.push\_back(std::make\_pair((\*i\_pt).first, (pos\_left\_line - lines.begin()) - (pos\_right\_line - part.begin())));

is\_prev = true;

prev\_pt = (\*i\_pt).second;

prev\_num = (pos\_left\_line - lines.begin()) - (pos\_right\_line - part.begin());

i\_pt++;

}

// Выводим количество найденных (оставшихся отрезков)

std::ostringstream oss;

// Сортируем массив точек

std::stable\_sort(numbers.begin(), numbers.end(), [](const std::pair<int, int> &e1, const std::pair<int, int> &e2) {return e1.first < e2.first;});

for (auto elem : numbers) { oss << elem.second << " "; }

std::cout << oss.str() << std::endl;

return 0;

}

# Задача 4: сортировка подсчетом

Первая строка содержит число 1≤*n*≤104, вторая — *n* натуральных чисел, не превышающих 10. Выведите упорядоченную по неубыванию последовательность этих чисел.

**Sample Input:**

5

2 3 9 2 9

**Sample Output:**

2 2 3 9 9

# Исходный код к задаче 4

#include <iostream>

#include <sstream>

#include <vector>

int main()

{

int n = 0;

std::cin >> n;

std::vector<unsigned int> b(11);

while (--n >= 0) {

unsigned int value = 0;

std::cin >> value;

b[value] += 1;

}

std::ostringstream oss;

for (unsigned int i = 0; i < 11; ++i)

for (unsigned int j = 0; j < b[i]; ++j) oss << i << " ";

std::cout << oss.str() << std::endl;

return 0;

}