**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЯ ОБРАЗОВАНИЯ**

**“ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”**

Факультет информационных технологий

Кафедра технологий программирования

**Лабораторная работа №1**

Выполнила студент 2 курса, группа 21-ИТ-1 Макеёнок Д.И.

Проверила Виноградова А.Д.

Полоцк, 2022 г.

**Модуль 2**

**Задание 1:** Дано целое число 1≤n≤40, необходимо вычислить n-е число Фибоначчи (напомним, что F0=0, F1=1 и Fn=Fn−1+Fn−2​ при n ≥ 2).

**Решение:**

#include <cassert>

#include <iostream>

class Fibonacci final {

public:

static int get(int n) {

assert(n >= 0);

int a = 0, b = 1;

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

a = a + b;

b = a - b;

}

return a;

}

};

int main(void) {

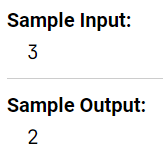
int n;

std::cin >> n;

std::cout << Fibonacci::get(n) << std::endl;

return 0;

}



**Задание 2:** Дано число 1≤n≤107, необходимо найти последнюю цифру n-го числа Фибоначчи.

**Решение:**

#include <cassert>

#include <iostream>

class Fibonacci final {

public:

static int get(int n) {

assert(n >= 0);

int a = 0, b = 1;

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

a = a + b;

b = a - b;

a = a % 10;

b = b % 10;

}

return a;

}

};

int main(void) {

int n;

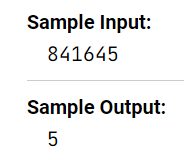
std::cin >> n;

std::cout << Fibonacci::get(n) << std::endl;

return 0;

}

}

****

**Задание 2:** По данным двум числам 1<=a, b<= 2\*10^9 найдите их наибольший общий делитель.

#include <cassert>

#include <cstdint>

#include <iostream>

template <class Int>

Int gcd(Int a, Int b) {

if (a == 0) {

return b;

}

else if (b == 0) {

return a;

}

else if (a >= b) {

return gcd(a % b, b);

}

else if (b >= a) {

return gcd(a, b % a);

}

}

int main(void) {

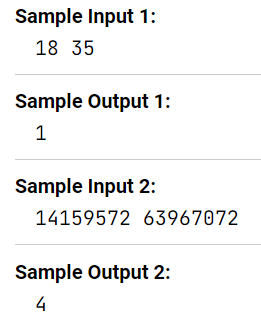
std::int32\_t a, b;

std::cin >> a >> b;

std::cout << gcd(a, b) << std::endl;

return 0;

}

****

**Модуль 4**

**Задание 1:** По данным n отрезкам необходимо найти множество точек минимального размера, для которого каждый из отрезков содержит хотя бы одну из точек.

**Решение:**

#include <iostream>

#include <vector>

#include <list>

using namespace std;

bool pred(pair<int, int> a, pair<int, int> b)

{

return a.second < b.second;

}

int main()

{

int count = 0;

cin >> count;

list<pair<int, int>> line;

for (size\_t i = 0; i < count; ++i) {

int a = 0, b = 0;

cin >> a >> b;

line.push\_back(make\_pair(a, b));

}

line.sort([](const pair<int, int>& a, const pair<int, int>& b) { return a.second < b.second; });

vector<int> points;

while (0 != line.size()) {

int p = (\*line.begin()).second;

points.push\_back(p);

while (true) {

if (line.size() != 0 && (\*line.begin()).first <= p) line.pop\_front();

else break;

}

}

size\_t pointscount = points.size();

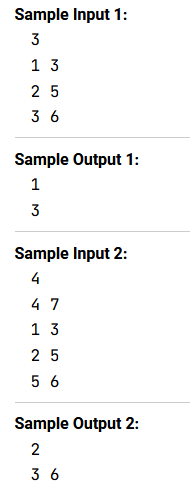
cout << pointscount << endl;

for (auto pt : points) cout << pt << " ";

cout << endl;

return 0;

}



**Задание 2:** Первая строка содержит количество предметов 1≤n≤103 и вместимость рюкзака 0≤W≤2⋅106. Каждая из следующих n строк задаёт стоимость 0≤ci≤2⋅106 и объём 0<wi≤2⋅106 предмета (n, W, ci​, wi​ — целые числа). Выведите максимальную стоимость частей предметов (от каждого предмета можно отделить любую часть, стоимость и объём при этом пропорционально уменьшатся), помещающихся в данный рюкзак, с точностью не менее трёх знаков после запятой.

**Решение:**

#include <iostream>

#include <iomanip>

#include <vector>

#include <algorithm>

using namespace std;

void fill\_section(vector<pair<double, double>>& v, const int n) {

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

double price, volume;

cin >> price >> volume;

v.push\_back(make\_pair(price, volume));

}

}

bool capacityeight\_cmp(const pair<double, double>& a, const pair<double, double>& b) {

return (a.first / a.second) > (b.first / b.second);

}

double calc\_cost(const pair<double, double>& c\_capacity) {

return c\_capacity.first / c\_capacity.second;

}

double fill\_knapsack(vector<pair<double, double>>& kn, int capacity) {

double sum\_cost = 0;

sort(kn.begin(), kn.end(), capacityeight\_cmp);

for (int i = 0; i < kn.size() && capacity > 0; ++i) {

auto item\_capacityeight = kn[i].second;

auto item\_cost = kn[i].first;

if (capacity >= item\_capacityeight) {

sum\_cost += item\_cost;

capacity -= item\_capacityeight;

}

else {

sum\_cost += calc\_cost(kn[i]) \* capacity;

capacity = 0;

}

}

return sum\_cost;

}

int main() {

int n, capacity;

vector<pair<double, double>> v;

cin >> n >> capacity;

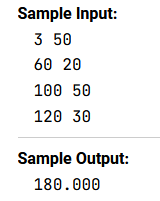
fill\_section(v, n);

auto sum = fill\_knapsack(v, capacity);

cout << fixed << setprecision(3) << sum;

return 0;

}



**Задание 3:** По данному числу 1≤n≤109 найдите максимальное число k, для которого n можно представить как сумму k различных натуральных слагаемых. Выведите в первой строке число k, во второй — k слагаемых.

**Решение:**

#include <iostream>

int main(void) {

int n, num = 1, sum = 0, mem[44720], counter = 0;

std::cin >> n;

if (n == 1) {

std::cout << 1 << std::endl << 1;

}

else if (n == 2) {

std::cout << 1 << std::endl << 2;

}

else {

for (int i = 0; true; i++) {

sum += num;

if (n - 2 \* (num + 1) >= sum) {

mem[i] = num;

num++;

counter++;

}

else {

if (num == n - sum) {

sum -= num;

mem[i] = n - sum;

counter++;

break;

}

else {

mem[i] = num;

mem[i + 1] = n - sum;

counter += 2;

break;

}

}

}

std::cout << counter << std::endl;

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

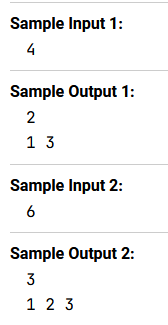
std::cout << mem[i] << " ";

}

}

return 0;

}



**Задание 4:** По данной непустой строке s длины не более 104, состоящей из строчных букв латинского алфавита, постройте оптимальный беспрефиксный код. В первой строке выведите количество различных букв k, встречающихся в строке, и размер получившейся закодированной строки. В следующих k строках запишите коды букв в формате "letter: code". В последней строке выведите закодированную строку.

**Решение:**

#include <iostream>

#include <string>

#include <unordered\_map>

#include <vector>

#include <queue>

class Huffman {

struct CharSetFrequency {

std::string char\_set;

int frequency;

bool operator < (const CharSetFrequency& other) const {

return std::tie(frequency, char\_set) > std::tie(other.frequency, other.char\_set);

}

};

public:

static std::unordered\_map<char, std::string> encode(const std::string& text);

static std::string decode(const std::string& text, const std::unordered\_map<char, std::string>& huffman\_encoding);

};

std::unordered\_map<char, std::string> Huffman::encode(const std::string& text) {

std::unordered\_map<char, int> char\_frequencies;

for (auto c : text) {

char\_frequencies[c]++;

}

std::vector<CharSetFrequency> frequencies;

for (auto char\_frequency : char\_frequencies) {

frequencies.push\_back({ std::string(1,char\_frequency.first), char\_frequency.second });

}

if (frequencies.size() == 1) {

std::unordered\_map<char, std::string> result;

result[frequencies[0].char\_set[0]] = "0";

return result;

}

std::unordered\_map<char, std::string> result;

std::priority\_queue<CharSetFrequency> q(frequencies.begin(), frequencies.end());

while (q.size() >= 2) {

auto first = q.top();

q.pop();

auto second = q.top();

q.pop();

for (auto c : first.char\_set) {

result[c] = "0" + result[c];

}

for (auto c : second.char\_set) {

result[c] = "1" + result[c];

}

q.push({ first.char\_set + second.char\_set, first.frequency + second.frequency });

}

return result;

}

std::string Huffman::decode(const std::string& text, const std::unordered\_map<char, std::string>& huffman\_encoding) {

size\_t len = text.size();

size\_t pos = 0;

std::string result;

while (pos < len) {

for (auto& encoded : huffman\_encoding) {

if (text.substr(pos, encoded.second.size()) == encoded.second) {

result += encoded.first;

pos += encoded.second.size();

break;

}

}

}

return result;

}

int main() {

std::string text;

std::cin >> text;

auto huffman\_encoding = Huffman::encode(text);

std::string encoded\_text;

for (auto c : text) {

encoded\_text += huffman\_encoding[c];

}

std::cout << huffman\_encoding.size() << " " << encoded\_text.size() << std::endl;

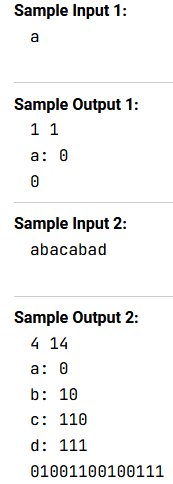
for (auto& encoded : huffman\_encoding) {

std::cout << encoded.first << ": " << encoded.second << std::endl;

}

std::cout << encoded\_text << std::endl;

}



**Задание 5:** Восстановите строку по её коду и беспрефиксному коду символов.

В первой строке входного файла заданы два целых числа k и l через пробел — количество различных букв, встречающихся в строке, и размер получившейся закодированной строки, соответственно. В следующих k строках записаны коды букв в формате "letter: code". Ни один код не является префиксом другого. Буквы могут быть перечислены в любом порядке. В качестве букв могут встречаться лишь строчные буквы латинского алфавита; каждая из этих букв встречается в строке хотя бы один раз. Наконец, в последней строке записана закодированная строка. Исходная строка и коды всех букв непусты. Заданный код таков, что закодированная строка имеет минимальный возможный размер.

В первой строке выходного файла выведите строку s. Она должна состоять из строчных букв латинского алфавита. Гарантируется, что длина правильного ответа не превосходит 104 символов.

**Решение:**

#include <iostream>

#include <map>

#include <string>

int main() {

int k, m;

std::cin >> k >> m;

std::map<std::string, char> dict;

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

{

std::string tmp1, tmp2;

std::cin >> tmp1 >> tmp2;

dict[tmp2] = tmp1[0];

}

std::string in;

std::cin >> in;

std::string tmp = "";

for (int i = 0; i < m; ++i)

{

tmp += in[i];

if (dict.find(tmp) != dict.end())

{

std::cout << dict[tmp];

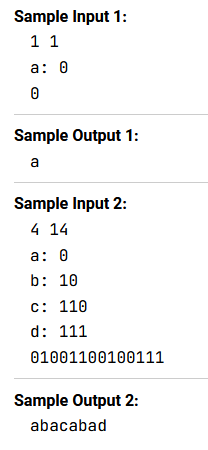
tmp = "";

}

}

return 0;

}



**Задание 6:** Первая строка входа содержит число операций 1≤n≤105. Каждая из последующих n строк задают операцию одного из следующих двух типов:

* Insert x, где 0 ≤ x ≤ 109 — целое число;
* ExtractMax.

Первая операция добавляет число x в очередь с приоритетами, вторая — извлекает максимальное число и выводит его.

**Решение:**

#include <iostream>

#include <string>

using namespace std;

struct Heap {

void siftup(int i) {

while (i > 0 && a[i / 2] < a[i]) {

swap(a[i], a[i / 2]);

i /= 2;

}

}

void insert(int v) {

n++;

a[n] = v;

siftup(n);

}

void siftdown(int i) {

while (2 \* i <= n) {

int j = i;

if (a[2 \* i] > a[j]) j = 2 \* i;

if (2 \* i + 1 <= n && a[2 \* i + 1] > a[j]) j = 2 \* i + 1;

if (j == i) break;

swap(a[i], a[j]);

i = j;

}

}

int ex\_max() {

int ex\_m = a[0];

a[0] = a[n];

n--;

siftdown(0);

return ex\_m;

}

int a[100000];

int n = -1;

};

int main() {

Heap h;

int n, num;

string s;

cin >> n;

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

cin >> s;

if (s == "Insert") {

cin >> num;

h.insert(num);

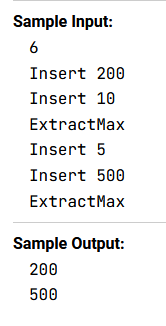
}

else cout << h.ex\_max() << endl;

}

return 0;

}



**Модуль 6**

**Задание 1:** 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 b1,…,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.

**Решение:**

#include <algorithm>

#include <cassert>

#include <iostream>

#include <vector>

int get\_pos(const std::vector<int>& numbers, int number) {

int l = 0;

int r = numbers.size() - 1;

while (l <= r) {

int m = (l + r) / 2;

if (numbers[m] == number)

return m + 1;

else if (numbers[m] > number)

r = m - 1;

else l = m + 1;

}

return -1;

}

int main(void) {

size\_t number\_count;

std::cin >> number\_count;

std::vector<int> numbers(number\_count);

for (auto& number : numbers) {

std::cin >> number;

}

assert(std::is\_sorted(numbers.begin(), numbers.end()));

size\_t query\_count;

std::cin >> query\_count;

while (query\_count-- > 0) {

int number;

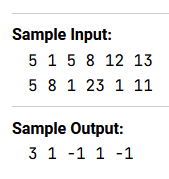
std::cin >> number;

std::cout << get\_pos(numbers, number) << " ";

}

std::cout << std::endl;

}

****

**Задание 2:** 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.)

**Решение:**

#include <vector>

#include <iomanip>

#include <iostream>

#include <ios>

using namespace std;

vector <long> merge(vector <long> a, vector<long> b, long& l) {

long inda = 0, indb = 0;

long sa = a.size(), sb = b.size();

vector <long> c(sa + sb);

for (long i = 0; i < sa + sb; i++) {

if (inda == sa) {

c[i] = b[indb];

indb++;

}

else if (indb == sb) {

c[i] = a[inda];

inda++;

}

else if (a[inda] <= b[indb]) {

c[i] = a[inda];

inda++;

}

else {

c[i] = b[indb];

indb++;

l += sa - inda;

}

}

return c;

}

int main()

{

long k, kol = 0;

cin >> k;

vector<vector<long>> Q(k);

vector <long> a(1);

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

cin >> a[0];

Q[i] = a;

}

long t = 0;

long S = Q.size();

while (S > 1) {

Q[t] = merge(Q[t], Q[t + 1], kol);

t += 2;

if (t >= S - 1) {

t = 0;

long y = 0;

while (2 \* y < S) {

Q[y] = Q[2 \* y];

y++;

}

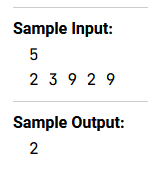
S = (S + 1) / 2;

}

}

cout << kol;

}



**Задание 3:** 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.

**Решение:**

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

int get\_pos(const std::vector<long long>& numbers, int number, bool mode\_l = true) {

int l = 0;

int r = numbers.size() - 1;

while (l <= r) {

int m = (l + r) / 2;

if (numbers[m] > number && mode\_l)

return m;

else if (numbers[m] > number && mode\_l == false)

return m;

else

l = m + 1;

}

return (l - 1);

}

int main() {

int n, m;

cin >> n >> m;

vector<long long> lefts(n);

vector<long long> rights(n);

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

cin >> lefts[i] >> rights[i];

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

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

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

int num, k1 = 0, k2 = 0;

cin >> num;

k1 = get\_pos(lefts, num);

while (k1 >= 0 && lefts[k1] > num)

k1--;

k2 = get\_pos(rights, num, false);

while (k2 >= 0 && rights[k2] >= num)

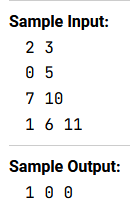
k2--;

cout << (k1 - k2) << ' ';

}

return 0;

}



**Задание 4:** Первая строка содержит число 1 ≤ n ≤ 104, вторая — n натуральных чисел, не превышающих 10. Выведите упорядоченную по неубыванию последовательность этих чисел.

**Решение:**

#include <iostream>

#include <vector>

std::vector<int> count\_sort(std::vector<int>& A) {

const int MAX\_NUM = 10000;

int B[MAX\_NUM + 1] = { 0 };

std::vector<int> result(A.size());

// Count freq

for (size\_t i = 0; i < A.size(); ++i) { ++B[A[i] - 1]; }

// Cumulative sum

for (size\_t i = 1; i <= MAX\_NUM; ++i) { B[i] += B[i - 1]; }

for (int i = A.size() - 1; i >= 0; --i) {

result[B[A[i] - 1] - 1] = A[i];

--B[A[i] - 1];

}

return result;

}

int main() {

int n;

std::cin >> n;

std::vector<int> A(n);

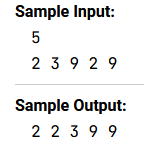
for (int i = 0; i < n; ++i) { std::cin >> A[i]; }

A = count\_sort(A);

for (int i = 0; i < n; ++i) { std::cout << A[i] << " "; }

return 0;

}



**Модуль 8**

**Задание 1:** Дано целое число 1≤*n*≤103 и массив  *A*[1…*n*] натуральных чисел, не превосходящих 2⋅109. Выведите максимальное  1≤*k*≤*n*, для которого найдётся подпоследовательность  1≤*i*1​<*i*2​<…<*ik*​≤*n* длины *k*, в которой каждый элемент делится на предыдущий (формально: для  всех 1≤*j*<*k*,  *A*[*ij*​]∣*A*[*ij*+1​]).

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main()

{

int n = 0;

cin >> n;

vector<int> arr;

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

{

int val = 0; cin >> val;

arr.push\_back(val);

}

auto size = arr.size();

vector<int> path\_len(size);

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

path\_len[i] = 1;

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

if ((arr[i] % arr[j] == 0) && (path\_len[j] + 1 > path\_len[i])) {

path\_len[i] = path\_len[j] + 1;

}

}

}

int max = 0;

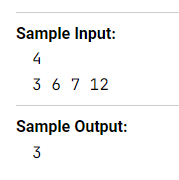
for (int i = 0; i < path\_len.size(); i++)

if (max < path\_len[i]) max = path\_len[i];

cout << max << endl;

return 0;

}



**Задание 2:** Вычислите расстояние редактирования двух данных непустых строк длины не более 102, содержащих строчные буквы латинского алфавита.

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

int main()

{

string str1, str2;

cin >> str1; cin >> str2;

int n = str1.size();

int m = str2.size();

vector<vector <int>> VE;

for (int i = 0; i <= n; i++) //inicializacia

VE.push\_back(vector<int>(m + 1));

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

VE[i][0] = i;

for (int i = 0; i <= m; ++i)

VE[0][i] = i;

int c = 0;

for (int i = 1; i <= n; i++) //sravnivanie

{

for (int j = 1; j <= m; j++)

{

if (str1[i - 1] == str2[j - 1]) c = 0;

else c = 1;

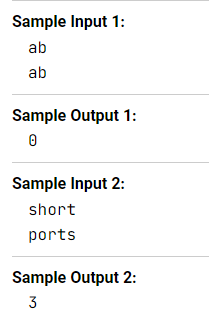
VE[i][j] = min({ VE[i - 1][j] + 1, VE[i][j - 1] + 1, VE[i - 1][j - 1] + c });

}

}

cout << VE[n][m] << endl;

}



**Задание 3:** Первая строка входа содержит целые числа 1≤*W*≤104 и 1≤*n*≤300 — вместимость рюкзака и число золотых слитков. Следующая строка содержит *n* целых чисел 0≤*w*1​,…,*wn*​≤105, задающих веса слитков. Найдите максимальный вес золота, который можно унести в рюкзаке.

#include <vector>

#include <iomanip>

#include <iostream>

using namespace std;

int main()

{

int W, n;

cin >> W >> n;

vector<int> R(n + 1);

for (int i = 1; i <= n; i++) cin >> R[i];

vector<vector<int>> D(W + 1, vector<int>(n + 1, 0));

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

for (int w = 1; w <= W; w++) {

D[w][i] = D[w][i - 1];

if (R[i] <= w) {

D[w][i] = max(D[w][i], D[w - R[i]][i - 1] + R[i]);

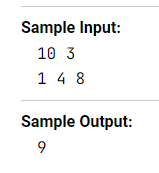
}

}

}

cout << D[W][n];

}



**Задание 4:** Даны число 1≤*n*≤102 ступенек лестницы и целые числа −104≤*a*1​,…,*an*​n≤104, которыми помечены ступеньки. Найдите максимальную сумму, которую можно получить, идя по лестнице снизу вверх (от нулевой до *n*-й ступеньки), каждый раз поднимаясь на одну или две ступеньки.

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

int main()

{

int steps;

cin >> steps;

vector<int> v;

v.push\_back(0);

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

int a = 0; cin >> a;

v.push\_back(a);

}

vector<int> arr;

arr.push\_back(0);

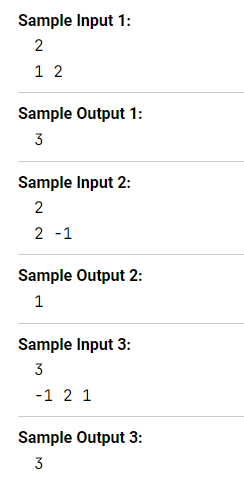
arr.push\_back(v[1]);

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

arr.push\_back(max({ arr[i - 1] + v[i], arr[i - 2] + v[i] }));

}

cout << arr[steps] << endl;

}

**Задание 5:** У вас есть примитивный калькулятор, который умеет выполнять всего три операции с текущим числом *x*: заменить *x* на 2*x*, 3*x* или *x*+1. По данному целому числу 1≤*n*≤105 определите минимальное число операций *k*, необходимое, чтобы получить *n* из 1. Выведите *k* и последовательность промежуточных чисел.

#include <iostream>

#include <vector>

#include <algorithm>

#include <string>

#include <sstream>

using namespace std;

int main()

{

int N = 0;

cin >> N;

vector<size\_t> S;

vector<vector<size\_t>> path(N + 1, vector<size\_t>());

S.push\_back(0);

S.push\_back(0);

path[1].push\_back(1);

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

size\_t x\_div\_3 = numeric\_limits<int>::max();

if ((i % 3) == 0) x\_div\_3 = S[i / 3] + 1;

size\_t x\_div\_2 = numeric\_limits<int>::max();

if ((i % 2) == 0) x\_div\_2 = S[i / 2] + 1;

size\_t min\_val = min({ x\_div\_3, x\_div\_2, S[i - 1] + 1 });

S.push\_back(min\_val);

if (min\_val == x\_div\_3) { path[i] = path[i / 3]; path[i].push\_back(i); }

if (min\_val == x\_div\_2) { path[i] = path[i / 2]; path[i].push\_back(i); }

if (min\_val == S[i - 1] + 1) { path[i] = path[i - 1]; path[i].push\_back(i); }

}

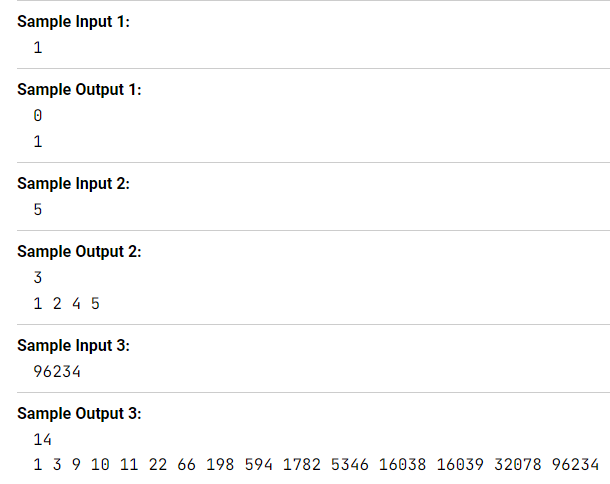
cout << S[N] << endl;

for (auto elem : path[N]) cout << elem << " ";

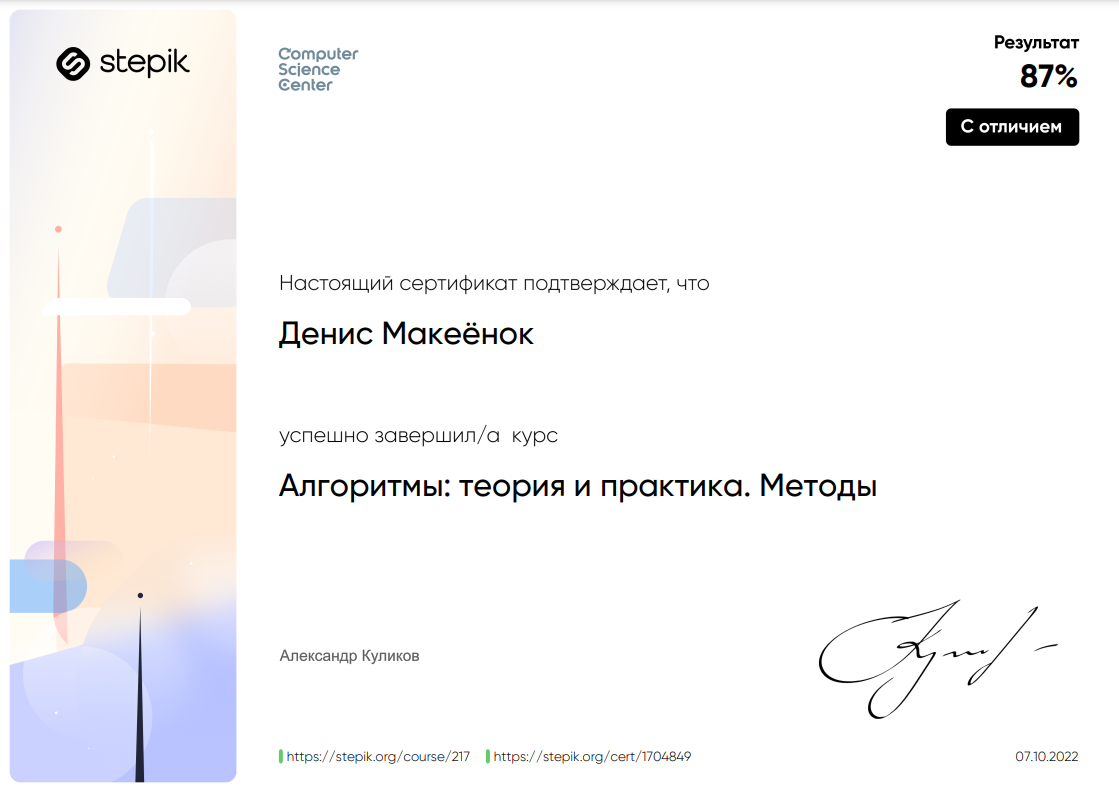
cout << endl;

return 0;

}



**Сертификат**

****