Отчет по дисциплине «Структуры и алгоритмы обработки данных»

Студента [скрыто], группы [скрыто]

Часть 1

Оглавление

[1 Обзор 3](#_Toc158548234)

[1.1 Пример задачи на программирование 3](#_Toc158548235)

[2 Введение 4](#_Toc158548236)

[2.2.6 Задача на программирование: небольшое число Фибоначчи 4](#_Toc158548237)

[2.2.7 Задача на программирование: последняя цифра большого числа Фибоначчи 4](#_Toc158548238)

[2.2.8 Задача на программирование повышенной сложности: огромное число Фибоначчи по модулю 4](#_Toc158548239)

[2.3.5 Задача на программирование: наибольший общий делитель 5](#_Toc158548240)

[2.4.7 Тест: правила работы с логарифмами 5](#_Toc158548241)

[2.4.8 Тест: правильная скорость роста 5](#_Toc158548242)

[2.4.9 Тест: правильная скорость роста 6](#_Toc158548243)

[2.4.10 Тест повышенной сложности: правильная скорость роста 6](#_Toc158548244)

[4 Жадные алгоритмы: теория и задачи 7](#_Toc158548245)

[4.1.9 Задача на программирование: покрыть отрезки точками 7](#_Toc158548246)

[4.1.10 Задача на программирование: непрерывный рюкзак 7](#_Toc158548247)

[4.1.11 Задача на программирование: различные слагаемые 8](#_Toc158548248)

[4.2.5 Задача на программирование: кодирование Хаффмана 9](#_Toc158548249)

[4.3.6 Тест: это куча? 12](#_Toc158548250)

[4.3.7 Тест: d-ичная куча 12](#_Toc158548251)

[6 «Разделяй и Властвуй»: теория и задачи 15](#_Toc158548252)

[6.1.4 Задача на программирование: двоичный поиск 15](#_Toc158548253)

[6.4.5 Задача на программирование: число инверсий 15](#_Toc158548254)

[6.8.3 Задача на программирование: сортировка подсчётом 17](#_Toc158548255)

[6.9.4 Тест повышенной сложности: нестандартные рекуррентные соотношения 19](#_Toc158548256)

[Сертификат 20](#_Toc158548257)

[[скрыто, сертификат 70%] 20](#_Toc158548258)

# 1 Обзор

## 1.1 Пример задачи на программирование

**Решение:**

#include <iostream>

int main**(){**

int a**,**b**;**

std**::**cin**>>**a**>>**b**;**

const int sum **=** a **+** b**;**

std**::**cout**<<**sum**;**

**return** 0**;**

**}**

# 2 Введение

## 2.2.6 Задача на программирование: небольшое число Фибоначчи

**Решение:**

#include <cstdint>

#include <iostream>

int main**(**void**)** **{**

std**::**uint\_fast32\_t n**;**

std**::**cin **>>** n**;**

std**::**uint\_fast32\_t prev**=**0**,**cur**=**1**,**next**=**1**;**

**while** **(**n**--){**

prev**=**cur**;**

cur**=**next**;**

next**=**cur**+**prev**;**

**}**

std**::**cout**<<**prev**;**

**return** 0**;**

**}**

## 2.2.7 Задача на программирование: последняя цифра большого числа Фибоначчи

**Решение:**

#include <cstdint>

#include <iostream>

int main**(**void**)** **{**

std**::**uint\_fast32\_t n**;**

std**::**cin **>>** n**;**

std**::**uint\_fast32\_t prev**=**0**,**cur**=**1**,**next**=**1**;**

**while** **(**n**--){**

prev**=**cur**;**

cur**=**next**;**

next**=(**cur**+**prev**)%**10**;**

**}**

std**::**cout**<<**prev**;**

**return** 0**;**

**}**

## 2.2.8 Задача на программирование повышенной сложности: огромное число Фибоначчи по модулю

**Решение:**

#include <cstdint>

#include <iostream>

int main**(**void**)** **{**

std**::**uint\_fast32\_t n**;**

std**::**cin **>>** n**;**

std**::**uint\_fast32\_t prev**=**0**,**cur**=**1**,**next**=**1**;**

**while** **(**n**--){**

prev**=**cur**;**

cur**=**next**;**

next**=(**cur**+**prev**)%**10**;**

**}**

std**::**cout**<<**prev**;**

**return** 0**;**

**}**

## 2.3.5 Задача на программирование: наибольший общий делитель

**Решение:**

#include <iostream>

#include <cstdint>

template **<**typename TyNum**>**

TyNum gcd**(**TyNum a**,** TyNum b**)**

**{**

**while(true)** **{**

**if(**a **>** b**)** **{**

**if(**b **==** 0**)**

**return** a**;**

a **%=** b**;**

**}** **else** **{**

**if(**a **==** 0**)**

**return** b**;**

b **%=** a**;**

**}**

**}**

**}**

int main**()** **{**

std**::**uint\_fast32\_t a**,** b**;**

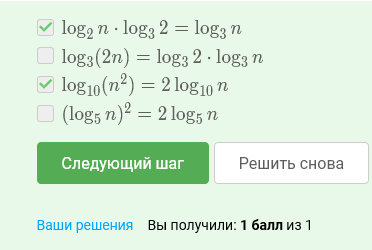
std**::**cin **>>** a **>>** b**;**

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

**return** 0**;**

**}**

## 2.4.7 Тест: правила работы с логарифмами



## 2.4.8 Тест: правильная скорость роста

## 2.4.9 Тест: правильная скорость роста

## 2.4.10 Тест повышенной сложности: правильная скорость роста



# 4 Жадные алгоритмы: теория и задачи

## 4.1.9 Задача на программирование: покрыть отрезки точками

**Решение:**

#include <cstddef>

#include <cstdint>

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

**using** pos\_type **=** std**::**int\_fast32\_t**;**

struct segment **{**

pos\_type min**,** max**;**

**};**

template **<**typename InputIt**,** typename Callback**>** void solve\_problem**(**InputIt first**,** InputIt last**,** Callback callback**)**

**{**

**if** **(**first**==**last**)** **return;**

segment border **{** **\***first **};**

**for(++**first**;** first **!=** last**;** **++**first**)** **{**

const segment**&** current **=** **\***first**;**

**if(**current**.**min **>=** border**.**min **&&** current**.**min **<=** border**.**max**)** **{**

border**.**min **=** current**.**min**;**

**if(**current**.**max **<** border**.**max**)**

border**.**max **=** current**.**max**;**

**}** **else** **{**

callback**(**border**.**min**);**

border **=** current**;**

**}**

**}**

callback**(**border**.**min**);**

**}**

int main**()**

**{**

std**::**size\_t count**;**

std**::**cin **>>** count**;**

std**::**vector**<**segment**>** segs**(**count**);**

std**::**for\_each**(**segs**.**begin**(),**segs**.**end**(),[](**segment**&** seg**){**std**::**cin **>>** seg**.**min **>>** seg**.**max**;});**

std**::**sort**(**segs**.**begin**(),** segs**.**end**(),** **[](**const segment**&** s1**,** const segment**&** s2**)** **{** **return** s1**.**min **<** s2**.**min**;** **});**

std**::**vector**<**pos\_type**>** result**;**

solve\_problem**(**segs**.**cbegin**(),** segs**.**cend**(),** **[&**result**](**pos\_type pos**)** **{** result**.**push\_back**(**pos**);** **});**

std**::**cout **<<** result**.**size**()** **<<** std**::**endl**;**

std**::**copy**(**result**.**begin**(),** result**.**end**(),** std**::**ostream\_iterator**<**pos\_type**>(**std**::**cout**,** " "**));**

**return** 0**;**

**}**

## 4.1.10 Задача на программирование: непрерывный рюкзак

**Решение:**

#include <algorithm>

#include <cstddef>

#include <cstdint>

#include <iomanip>

#include <iostream>

#include <vector>

struct item **{**

std**::**uint\_fast32\_t cost**;**

std**::**uint\_fast32\_t size**;**

**};**

template **<**typename InputIt**,** typename BackpackTy**>**

double solve\_problem**(**InputIt first**,** InputIt last**,** BackpackTy backpack\_size**)**

**{**

double max\_cost **=** 0.0**;**

**for(;** first **!=** last**;** **++**first**)** **{**

const auto**&** item **=** **\***first**;**

**if(**item**.**size **<=** backpack\_size**)** **{**

max\_cost **+=** item**.**cost**;**

backpack\_size **-=** item**.**size**;**

**}** **else** **{**

const double relative\_cost**=**item**.**cost **/** **static\_cast<**double**>(**item**.**size**);**

max\_cost **+=** relative\_cost **\*** backpack\_size**;**

**break;**

**}**

**}**

**return** max\_cost**;**

**}**

int main**()**

**{**

std**::**size\_t count**;**

std**::**uint\_fast32\_t backpack\_size**;**

std**::**cin **>>** count **>>** backpack\_size**;**

std**::**vector**<**item**>** items**(**count**);**

std**::**for\_each**(**items**.**begin**(),** items**.**end**(),** **[](**item**&** item**)** **{**std**::**cin **>>** item**.**cost **>>** item**.**size**;});**

std**::**sort**(**items**.**begin**(),** items**.**end**(),** **[](**const item**&** lhs**,** const item**&** rhs**)** **{**

**return** lhs**.**cost **/** **static\_cast<**double**>(**lhs**.**size**)** **>** rhs**.**cost **/** **static\_cast<**double**>(**rhs**.**size**);**

**});**

std**::**cout **<<** std**::**fixed **<<** std**::**setprecision**(**3**)** **<<** solve\_problem**(**items**.**begin**(),** items**.**end**(),** backpack\_size**);**

**return** 0**;**

**}**

## 4.1.11 Задача на программирование: различные слагаемые

**Решение:**

#include <iostream>

#include <cstdint>

#include <cmath>

**using** number\_t**=**std**::**uint\_fast32\_t**;**

int main**()** **{**

number\_t n**;**

std**::**cin**>>**n**;**

//(n^2+n)/2 - известная закономерность для расчёта суммы первых n чисел

number\_t terms**=static\_cast<**number\_t**>(**std**::**floor**((**std**::**sqrt**(**8.0**\***n**+**1**)-**1**)\***0.5**));**

std**::**cout**<<**terms**<<**std**::**endl**;**

**for** **(**number\_t i**=**1**;**i**<**terms**;++**i**){**

std**::**cout**<<**i**<<**' '**;**

**}**

**--**terms**;**

std**::**cout**<<**n**-(**terms**\***terms**+**terms**)/**2**;**

**return** 0**;**

**}**

## 4.2.5 Задача на программирование: кодирование Хаффмана

**Решение:**

#include <algorithm>

#include <array>

#include <cstddef>

#include <cstdint>

#include <deque>

#include <fstream>

#include <iostream>

#include <iterator>

#include <memory>

#include <string>

#include <unordered\_map>

#include <limits>

**using** byte **=** unsigned char**;**

struct continue\_node**;**

struct leaf\_node**;**

struct node**;**

struct visitor **{**

virtual void visit\_continue\_node**(**const continue\_node**&** node**)** **=** 0**;**

virtual void visit\_leaf\_node**(**const leaf\_node**&** node**)** **=** 0**;**

virtual void begin\_visit**(**const node**&** node**)** **=** 0**;**

**};**

struct node **{**

std**::**size\_t weight**;**

node**(**std**::**size\_t weight**):**weight**(**weight**){}**

virtual **~**node**()** **=** **default;**

virtual void visit**(**visitor**&** visitor**)** const **=** 0**;**

**};**

struct continue\_node **:** public node **{**

std**::**unique\_ptr**<**node**>** left**,** right**;**

continue\_node**(**std**::**size\_t weight**,** std**::**unique\_ptr**<**node**>** left**,** std**::**unique\_ptr**<**node**>** right**)**

**:** node**(**weight**),** left**(**std**::**move**(**left**)),** right**(**std**::**move**(**right**)){}**

void visit**(**visitor**&** visitor**)** const override

**{**

visitor**.**visit\_continue\_node**(\*this);**

**}**

**};**

struct leaf\_node **:** public node **{**

byte value**;**

leaf\_node**(**std**::**size\_t weight**,** byte value**):** node**(**weight**),** value**(**value**){}**

void visit**(**visitor**&** visitor**)** const override

**{**

visitor**.**visit\_leaf\_node**(\*this);**

**}**

**};**

struct visitor256bet **:** public visitor **{**

public**:**

struct code**{**

unsigned int value**;**

unsigned int length**;**

static std**::**string format\_code**(**code code**){**

std**::**string res**;**

**while(**code**.**length**--)** **{**

res**.**push\_back**((**code**.**value **&** 1**)** **==** 1 **?** '1' **:** '0'**);**

code**.**value **>>=** 1**;**

**};**

**return** res**;**

**}**

**};**

private**:**

code cur\_code**;**

std**::**array**<**code**,** std**::**numeric\_limits**<**byte**>::**max**()>** table**;**

public**:**

void visit\_continue\_node**(**const continue\_node**&** node**)** override

**{**

**++**cur\_code**.**length**;**

const auto mask **=** 1 **<<** **(**cur\_code**.**length **-** 1**);**

cur\_code**.**value **&=** **~**mask**;**

node**.**left**->**visit**(\*this);**

cur\_code**.**value **|=** mask**;**

node**.**right**->**visit**(\*this);**

**--**cur\_code**.**length**;**

**}**

void visit\_leaf\_node**(**const leaf\_node**&** node**)** override

**{**

table**[**node**.**value**]** **=** cur\_code**;**

**}**

void begin\_visit**(**const node**&** node**)** override

**{**

table**.**fill**(**code**{**0**,**0**});**

cur\_code**=**code**{**0**,dynamic\_cast<**const leaf\_node**\*>(&**node**)** **!=** **nullptr** **?** 1 **:** 0**};**

node**.**visit**(\*this);**

**}**

template **<**typename InputIt**,** typename Callback**>** void encode**(**InputIt first**,** InputIt last**,** Callback callback**)**

**{**

**for(;** first **!=** last**;** **++**first**)** **{**

callback**(**table**[(**unsigned char**)(\***first**)]);**

**}**

**}**

template **<**typename Callback**>**

void get\_table**(**Callback callback**){**

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

code cd **=** table**[static\_cast<**byte**>(**i**)];**

**if** **(**cd**.**length**!=**0**)** callback**(**cd**,**i**);}**

**}**

**};**

template **<**typename InputIt**>** std**::**unordered\_map**<**byte**,** std**::**size\_t**>** get\_freq\_map**(**InputIt first**,** InputIt last**)**

**{**

std**::**unordered\_map**<**byte**,** std**::**size\_t**>** map**;**

std**::**for\_each**(**first**,** last**,** **[&**map**](**const byte ch**)** **{** **++**map**[**ch**];** **});**

**return** map**;**

**}**

template **<**typename FreqMapIt**>** std**::**unique\_ptr**<**node**>** build\_huffman\_tree**(**FreqMapIt first**,** FreqMapIt last**)**

**{**

std**::**deque**<**std**::**unique\_ptr**<**leaf\_node**>>** leafs**;**

std**::**for\_each**(**first**,** last**,**

**[&**leafs**](**const auto**&** m**)** **{** leafs**.**push\_back**(**std**::**make\_unique**<**leaf\_node**>(**m**.**second**,** **(**byte**)(**m**.**first**)));** **});**

**switch(**leafs**.**size**())** **{**

**case** 0**:**

**return** **nullptr;**

**case** 1**:**

**return** std**::**move**(**leafs**.**front**());**

**default:**

**break;**

**}**

std**::**deque**<**std**::**unique\_ptr**<**continue\_node**>>** trees**;**

std**::**sort**(**leafs**.**begin**(),** leafs**.**end**(),**

**[](**const std**::**unique\_ptr**<**leaf\_node**>&** l**,** const std**::**unique\_ptr**<**leaf\_node**>&** r**)** **{** **return** l**->**weight **>** r**->**weight**;** **});**

std**::**array**<**std**::**unique\_ptr**<**node**>** **,**2**>** nodes**;**

**while(!**leafs**.**empty**()** **||** trees**.**size**()** **>** 1**)** **{**

**for(**auto**&** node**:**nodes**)** **{**

**if(**trees**.**empty**()** **||** **(!**leafs**.**empty**()** **&&** leafs**.**back**()->**weight **<=** trees**.**back**()->**weight**))** **{**

node **=** std**::**move**(**leafs**.**back**());**

leafs**.**pop\_back**();**

**}** **else** **{**

node **=** std**::**move**(**trees**.**back**());**

trees**.**pop\_back**();**

**}**

**}**

auto parent **=** std**::**make\_unique**<**continue\_node**>(**

nodes**[**0**]->**weight **+** nodes**[**1**]->**weight**,** std**::**move**(**nodes**[**0**]),** std**::**move**(**nodes**[**1**]));**

trees**.**push\_front**(**std**::**move**(**parent**));**

**}**

**return** std**::**move**(**trees**.**front**());**

**}**

int main**()**

**{**

std**::**ios**::**sync\_with\_stdio**(false);**

std**::**string input**;**

/\*из-за особенностей проверяющей системы пришлось использовать std::istream\_iterator,

хотя предполгагалось std::istreambuf\_iterator\*/

std**::**copy**(**std**::**istream\_iterator**<**char**>(**std**::**cin**),** std**::**istream\_iterator**<**char**>(),**std**::**back\_inserter**(**input**));**

std**::**unordered\_map**<**byte**,** std**::**size\_t**>** map **=** get\_freq\_map**(**input**.**cbegin**(),** input**.**cend**());**

std**::**unique\_ptr**<**node**>** root**(**build\_huffman\_tree**(**map**.**cbegin**(),** map**.**cend**()));**

**if(**root **==** **nullptr)**

**return** 1**;**

visitor256bet vis**;**

vis**.**begin\_visit**(\***root**);**

std**::**string encoded\_str**;**

vis**.**encode**(**input**.**cbegin**(),** input**.**cend**(),** **[&**encoded\_str**](**const visitor256bet**::**code**&** code**)** **{**

encoded\_str**.**append**(**visitor256bet**::**code**::**format\_code**(**code**));});**

std**::**cout**<<**map**.**size**()<<**' '**<<**encoded\_str**.**size**()<<**std**::**endl**;**

vis**.**get\_table**([](**const visitor256bet**::**code**&** code**,** byte index**){**

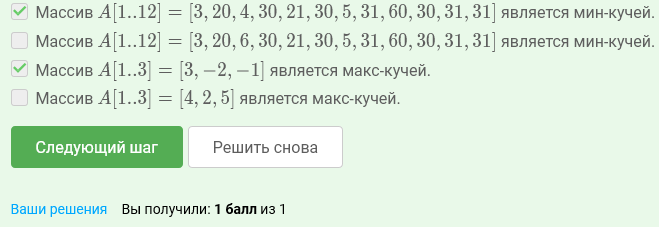
std**::**cout**<<static\_cast<**char**>(**index**)<<**": "**<<**visitor256bet**::**code**::**format\_code**(**code**)<<** '\n'**;});**

std**::**cout**<<**encoded\_str**;**

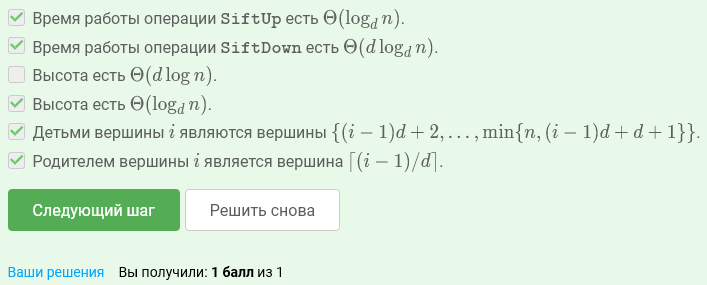
**return** 0**;**

**}**

## 4.3.6 Тест: это куча?



## 4.3.7 Тест: d-ичная куча

4.3.8 Задача на программирование: очередь с приоритетами

**Решение:**

#include <cstddef>

#include <iostream>

#include <string>

#include <utility>

#include <vector>

#include <cstdint>

**using** el\_ty**=**int\_fast32\_t**;**

std**::**vector**<**el\_ty**>** queue**;**

void insert**(**const el\_ty val**)**

**{**

queue**.**push\_back**(**val**);**

const std**::**size\_t size **=** queue**.**size**();**

**if(**size **==** 1**)**

**return;**

std**::**size\_t cur\_ind **=** size **-** 1**;**

std**::**size\_t parent\_ind **=** size **/** 2 **-** 1**;** //(cur\_ind + 1)/2-1

**while(**queue**[**cur\_ind**]** **>** queue**[**parent\_ind**])** **{**

std**::**swap**(**queue**[**cur\_ind**],** queue**[**parent\_ind**]);**

**if(**parent\_ind **==** 0**)**

**break;**

cur\_ind **=** parent\_ind**;**

parent\_ind **=** **(**cur\_ind **+** 1**)** **/** 2 **-** 1**;**

**}**

**}**

el\_ty extract\_max**()**

**{**

const el\_ty res **=** queue**[**0**];**

queue**[**0**]** **=** queue**.**back**();**

queue**.**pop\_back**();**

const std**::**size\_t size **=** queue**.**size**();**

std**::**size\_t cur\_ind **=** 0**;**

**while(true)** **{**

const std**::**size\_t child\_ind1 **=** cur\_ind **\*** 2 **+** 1**;**

const std**::**size\_t child\_ind2 **=** child\_ind1 **+** 1**;**

std**::**size\_t max\_ind**;**

**if(**child\_ind1 **<** size **&&** child\_ind2 **<** size**)**

max\_ind **=** queue**[**child\_ind1**]** **>=** queue**[**child\_ind2**]** **?** child\_ind1 **:** child\_ind2**;**

**else** **{**

**if(**child\_ind1 **<** size**)**

max\_ind **=** child\_ind1**;**

**else**

**break;**

**}**

**if(**queue**[**max\_ind**]** **>** queue**[**cur\_ind**])** **{**

std**::**swap**(**queue**[**cur\_ind**],** queue**[**max\_ind**]);**

cur\_ind **=** max\_ind**;**

**}** **else**

**break;**

**}**

**return** res**;**

**}**

int main**()**

**{**

std**::**size\_t operations**;**

std**::**cin **>>** operations**;**

queue**.**reserve**(**operations **/** 2**);**

std**::**string name**;**

el\_ty val**;**

**while(**operations**--)** **{**

std**::**cin **>>** name**;**

**if(**name **==** "Insert"**)** **{**

std**::**cin **>>** val**;**

insert**(**val**);**

**}** **else** **if(**name **==** "ExtractMax"**)** **{**

std**::**cout **<<** extract\_max**()** **<<** std**::**endl**;**

**}**

**}**

**}**

# 6 «Разделяй и Властвуй»: теория и задачи

## 6.1.4 Задача на программирование: двоичный поиск

**Решение:**

#include <algorithm>

#include <cstddef>

#include <cstdint>

#include <iostream>

#include <iterator>

#include <vector>

**namespace** local**{**

template **<**typename InputIt**,** typename ElTy**>** InputIt binary\_search**(**InputIt first**,** InputIt last**,** const ElTy**&** target**)**

**{**

typename std**::**iterator\_traits**<**InputIt**>::**difference\_type count **=** std**::**distance**(**first**,** last**);**

InputIt temp\_it**;**

**while(**count **>** 0**)** **{**

temp\_it **=** first**;**

const bool odd **=** count **&** 1**;**

count **/=** 2**;**

std**::**advance**(**temp\_it**,** count **-** **!**odd**);**

**if(**target **>** **\***temp\_it**)** **{**

first **=** **++**temp\_it**;**

**}** **else** **if(**target **==** **\***temp\_it**)** **{**

**return** temp\_it**;**

**}**

**}**

**return** last**;**

**}}**

**using** number\_t **=** std**::**int\_fast32\_t**;**

int main**()**

**{**

std**::**size\_t count**;**

std**::**cin **>>** count**;**

std**::**vector**<**number\_t**>** numbers**(**count**);**

std**::**for\_each**(**numbers**.**begin**(),** numbers**.**end**(),** **[](**number\_t**&** num**)** **{** std**::**cin **>>** num**;** **});**

std**::**size\_t search\_count**;**

std**::**cin **>>** search\_count**;**

**while(**search\_count**--)** **{**

number\_t val**;**

std**::**cin **>>** val**;**

auto it **=** local**::**binary\_search**(**numbers**.**cbegin**(),** numbers**.**cend**(),** val**);**

**if(**it **!=** numbers**.**cend**())**

std**::**cout **<<** std**::**distance**(**numbers**.**cbegin**(),**it**)+**1 **<<** ' '**;**

**else**

std**::**cout **<<** **-**1 **<<** ' '**;**

**}**

**}**

## 6.4.5 Задача на программирование: число инверсий

**Решение:**

#include <iostream>

#include <vector>

#include <iterator>

#include <algorithm>

#include <cstddef>

#include <cstdint>

**namespace** \_merge\_sort\_implementation\_details

**{**

template **<**bool count\_inversions**>** struct inversions **{**

**using** count\_t**=**std**::**size\_t**;**

static count\_t count**;**

**};**

template **<>** struct inversions**<false>** **{};**

template**<>** inversions**<true>::**count\_t inversions**<true>::**count**=**0**;**

#pragma push\_macro("if\_constexpr")

#ifdef if\_constexpr

#undef if\_constexpr

#endif

#if \_\_cplusplus>=201703L

#define if\_constexpr if constexpr

#else

#define if\_constexpr if

#endif

template **<**bool count\_inversions**>** struct impl **{**

template **<**typename LeftIt**,** typename RightIt**,** typename BufferIt**>**

static void merge**(**LeftIt left**,** LeftIt left\_last**,** RightIt right**,** RightIt right\_last**,** BufferIt buffer**)**

**{**

**for(;** left **!=** left\_last **&&** right **!=** right\_last**;** **++**buffer**)** **{**

const typename std**::**iterator\_traits**<**LeftIt**>::**value\_type**&** l\_val **=** **\***left**;**

const typename std**::**iterator\_traits**<**RightIt**>::**value\_type**&** r\_val **=** **\***right**;**

**if(**r\_val **<** l\_val**)** **{**

**\***buffer **=** r\_val**;**

**++**right**;**

if\_constexpr**(**count\_inversions**)**

inversions**<true>::**count **+=** std**::**distance**(**left**,** left\_last**);**

**}** **else** **{**

**\***buffer **=** l\_val**;**

**++**left**;**

**}**

**}**

std**::**copy**(**left**,** left\_last**,** buffer**);**

std**::**copy**(**right**,** right\_last**,** buffer**);**

**}**

template **<**typename DataIt**,** typename BufIt**>**

static void merge\_step**(**DataIt data\_it**,**

DataIt data\_end**,**

BufIt buf\_it**,**

BufIt buf\_end**,**

const std**::**size\_t length**)** // length should be power of two

**{**

const std**::**size\_t half\_length **=** length **/** 2**;**

**for(;** buf\_end **-** buf\_it **>=** length**;** buf\_it **+=** length**,** data\_it **+=** length**)** **{**

merge**(**data\_it**,** data\_it **+** half\_length**,** data\_it **+** half\_length**,** data\_it **+** length**,** buf\_it**);**

**}**

**if(**buf\_it **!=** buf\_end**)** **{**

buf\_it **-=** length**;**

DataIt merging\_it **=** data\_it **-** length**;**

merge**(**buf\_it**,** buf\_it **+** length**,** data\_it**,** data\_end**,** merging\_it**);**

std**::**copy**(**merging\_it**,** data\_end**,** buf\_it**);**

**}**

**}**

template **<**typename InputIt**>** static void merge\_sort**(**InputIt first**,** InputIt last**)**

**{**

if\_constexpr**(**count\_inversions**)**

inversions**<true>::**count **=** 0**;**

std**::**vector**<**typename std**::**iterator\_traits**<**InputIt**>::**value\_type**>** buffer**(**std**::**distance**(**first**,** last**));**

const std**::**size\_t size **=** buffer**.**size**();**

const auto buf\_start **=** buffer**.**begin**();**

const auto buf\_end **=** buffer**.**end**();**

std**::**size\_t step **=** 1**;**

**while(**step **\*** 4 **<=** size**)** **{**

merge\_step**(**first**,** last**,** buf\_start**,** buf\_end**,** step **\*** 2**);**

merge\_step**(**buf\_start**,** buf\_end**,** first**,** last**,** step **\*** 4**);**

step **\*=** 4**;**

**}**

**if(**step **\*** 2 **<=** size**)** **{**

merge\_step**(**first**,** last**,** buf\_start**,** buf\_end**,** step **\*** 2**);**

**if(**step **\*** 2 **==** size**)**

std**::**copy**(**buffer**.**cbegin**(),** buffer**.**cend**(),** first**);**

**}**

**}**

**};**

#undef if\_constexpr

#pragma pop\_macro("if\_constexpr")

**}**

template **<**typename InputIt**>** void merge\_sort**(**InputIt first**,** InputIt last**)**

**{**

\_merge\_sort\_implementation\_details**::**impl**<false>::**merge\_sort**(**first**,** last**);**

**}**

template **<**typename InputIt**>** \_merge\_sort\_implementation\_details**::**inversions**<true>::**count\_t merge\_sort\_inversions**(**InputIt first**,** InputIt last**)**

**{**

\_merge\_sort\_implementation\_details**::**impl**<true>::**merge\_sort**(**first**,** last**);**

**return** \_merge\_sort\_implementation\_details**::**inversions**<true>::**count**;**

**}**

**using** number\_t**=**std**::**int\_fast32\_t**;**

int main**()** **{**

std**::**size\_t count**;**

std**::**cin**>>**count**;**

std**::**vector**<**number\_t**>** data**(**count**);**

std**::**for\_each**(**data**.**begin**(),**data**.**end**(),[](**number\_t**&** elem**){**std**::**cin**>>**elem**;});**

std**::**cout**<<**merge\_sort\_inversions**(**data**.**begin**(),**data**.**end**());**

**return** 0**;**

**}**

## 6.8.3 Задача на программирование: сортировка подсчётом

**Решение:**

#include <algorithm>

#include <iostream>

#include <vector>

#include <cstddef>

#include <type\_traits>

#include <iterator>

template **<**typename InputIt**,** class WightFunc**>**

void count\_sort**(**InputIt first**,** InputIt last**,** WightFunc weight\_func**,** std**::**size\_t weight\_difference**)**

**{**

**if(**first **==** last**)**

**return;**

std**::**vector**<**typename std**::**iterator\_traits**<**InputIt**>::**value\_type**>** result**((**std**::**size\_t**)**std**::**distance**(**first**,** last**));**

std**::**vector**<**std**::**size\_t**>** reps**(**weight\_difference **+** 1**);**

**for(**auto it **=** first**;** it **!=** last**;** **++**it**)** **{**

**++**reps**[(**std**::**size\_t**)**weight\_func**(\***it**)];**

**}**

std**::**size\_t prev **=** reps**.**front**();**

reps**.**front**()** **=** 0**;**

**for(**auto it **=** reps**.**begin**()** **+** 1**;** it **!=** reps**.**end**();** **++**it**)** **{**

const std**::**size\_t orig **=** **\***it**;**

**\***it **=** prev **+** **\*(**it **-** 1**);**

prev **=** orig**;**

**}**

**for(**auto it **=** first**;** it **!=** last**;** **++**it**)** **{**

result**[**reps**[(**std**::**size\_t**)**weight\_func**(\***it**)]++]** **=** std**::**move**(\***it**);**

**}**

std**::**move**(**result**.**cbegin**(),** result**.**cend**(),** first**);**

**}**

template **<**typename InputIt**,** typename **=** typename std**::**enable\_if**<**std**::**is\_integral**<**typename std**::**iterator\_traits**<**InputIt**>::**value\_type**>::**value**>::**type**>**

void count\_sort**(**InputIt first**,** InputIt last**)**

**{**

//const auto [min, max] = std::minmax\_element(first, last);

**using** val\_t**=**typename std**::**iterator\_traits**<**InputIt**>::**value\_type**;**

const auto min\_max **=** std**::**minmax\_element**(**first**,** last**);**

count\_sort**(**

first**,** last**,** **[&**min\_max**](**val\_t item**)** **{** **return** **static\_cast<**std**::**size\_t**>(**item **-** **\***min\_max**.**first**);** **},**

**static\_cast<**std**::**size\_t**>(\***min\_max**.**second **-** **\***min\_max**.**first**));**

**}**

**using** number\_t**=**int**;**

int main**()** **{**

std**::**size\_t count**;**

std**::**cin**>>**count**;**

std**::**vector**<**number\_t**>** data**(**count**);**

std**::**for\_each**(**data**.**begin**(),**data**.**end**(),[](**number\_t**&** elem**){**std**::**cin**>>**elem**;});**

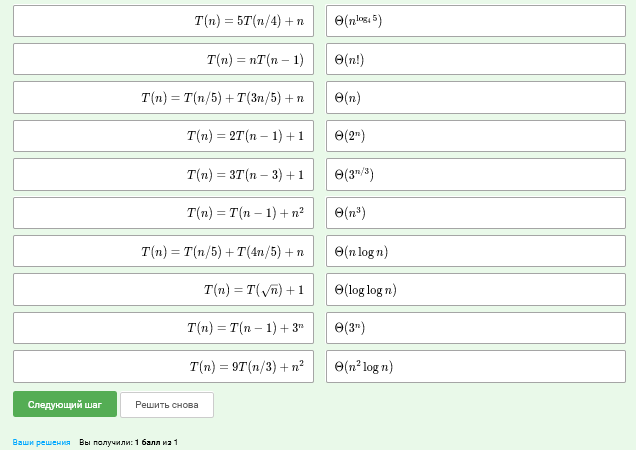
count\_sort**(**data**.**begin**(),**data**.**end**());**

std**::**copy**(**data**.**begin**(),**data**.**end**(),** std**::**ostream\_iterator**<**number\_t**>(**std**::**cout**,** " "**));**

**return** 0**;**

**}**

## 6.9.4 Тест повышенной сложности: нестандартные рекуррентные соотношения



# Сертификат

## [скрыто, сертификат 70%]