

Национальный исследовательский университет  
«Высшая Школа Экономики»  
Московский институт электроники и математики им. А. Н. Тихонова

**Лабораторная работа №1**

Выполнил:  
Дёма Иван Романович,  
СКБ212

Проверил:  
Драчёв Григорий Александрович

Москва, 2024

## 1. Задание

Реализовать сортировки на примере некоторого класса.

Выбрать 7-10 наборов данных для сортировки засечь время каждой из них.

Для варианты номер 9 необходимо реализовать сортировки:

- А) Пузырьком
- Б) Шейкер-сортировку
- В) Слиянием

В качестве класса использовать массив данных об экспортируемых товарах, состоящий из имени товаров, страны экспорта, количества партии и цены партии.

## 2. Сравнение сортировок

Для сравнения сортировок были построены графики с применением средств визуализации matplotlib в python в обычной и логарифмической шкале.

График 1:

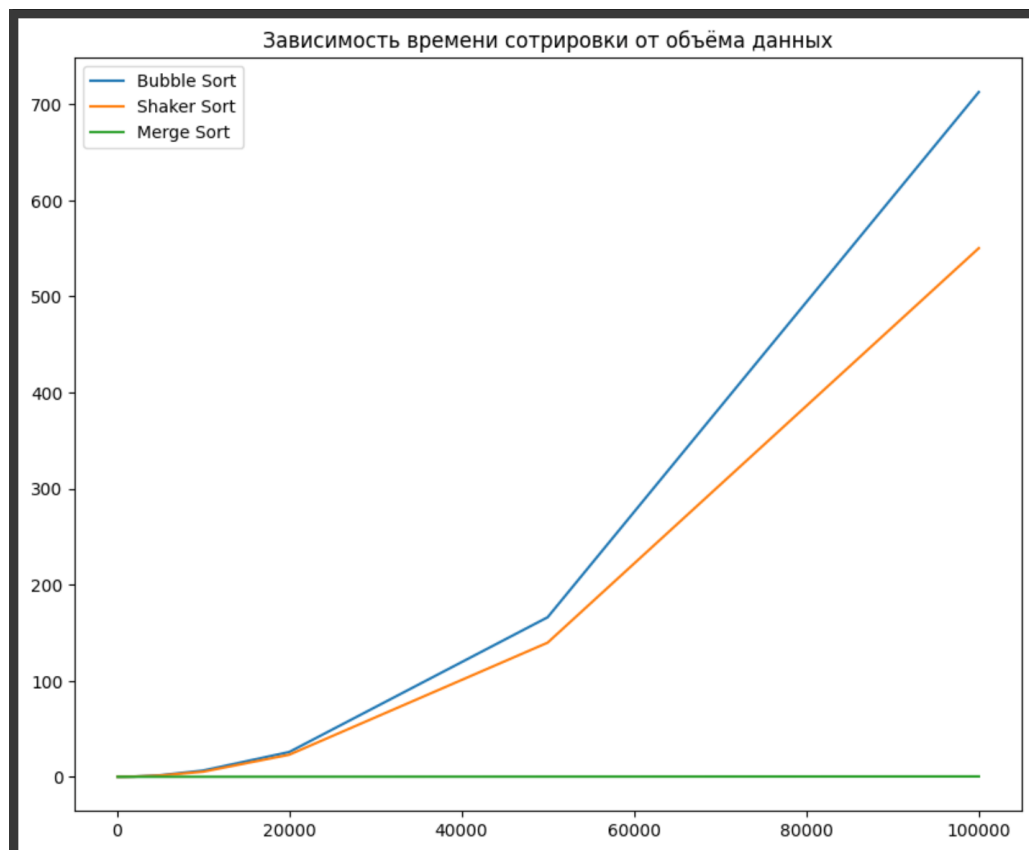
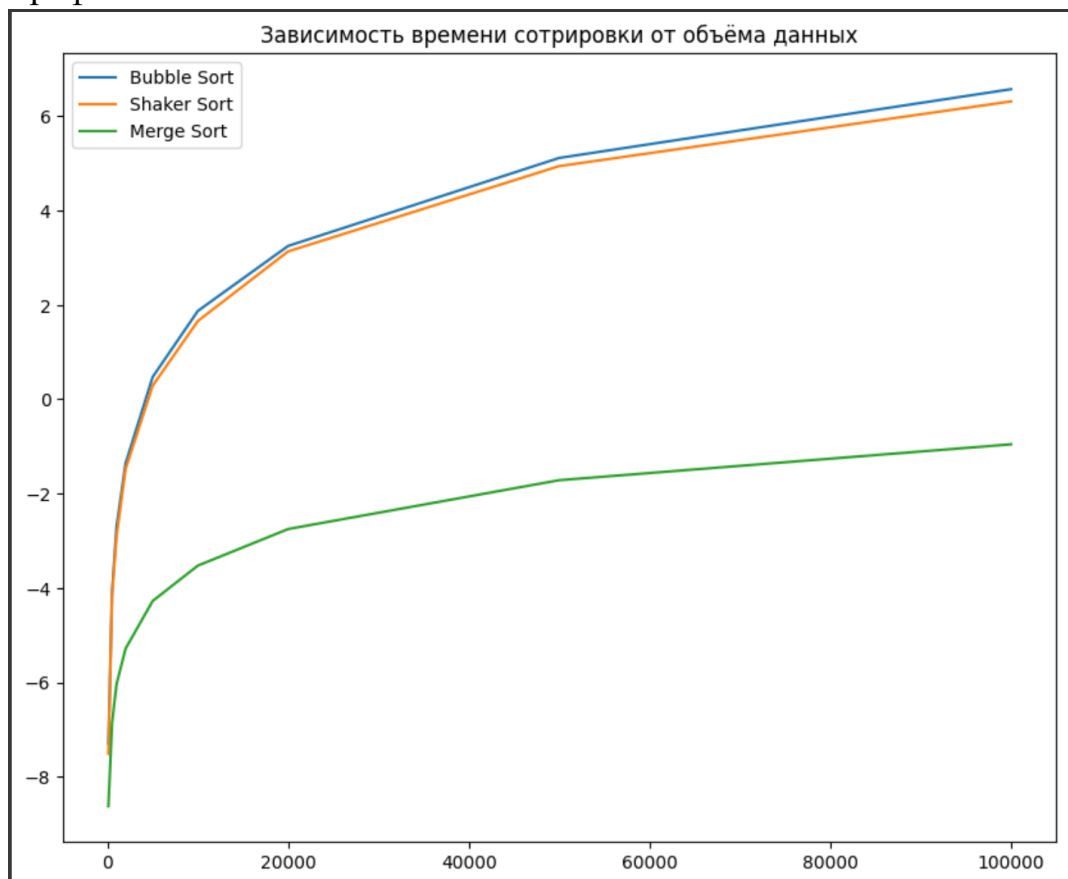


График 2:



### 3. Выводы

**Bubble sort** – самая не эффективная, но простая для написания и удобная в некоторых случаях сортировка. Во всех случаях имеет сложность  $O(n^2)$ . Не требует дополнительной памяти. Обладает устойчивостью.

Удобная для небольших массивов, которые нужно быстро и без мороки отсортировать. Очень эффективна, в случае, когда заранее известно, что часть элементов (стоящая левее) практически отсортирована.

**Shaker sort** – улучшенная версия пузырьковой сортировки, более универсальная и не менее удобная. Имеет сложность  $O(n^2)$ . Не требует дополнительной памяти. Обладает устойчивостью.

Более универсальная сортировка, но при этом эффективна только на массивах небольшой длины (или на уже частично отсортированных)

**Merge sort** – достаточно быстрая и эффективная сортировка. Позволяет быстро отсортировать данные, если практически ничего не известно о структуре данных. Во всех случаях имеет сложность  $O(n \log n)$ . Использует дополнительную память (равную размеру массива). Обладает устойчивостью.

Самая эффективная сортировка из предложенных, подходящая для массивов больших размеров. Может стать палочкой-выручалочкой, когда нет возможности выгрузить весь массив в оперативную память. Подходит, когда в программе используется распараллеливание.

## 4. Документацию кода

Репозиторий:

### Список файлов

#### Файлы

Полный список файлов.

<b>exportGoods.cpp</b>	910
<b>exportGoods.h</b>	1011
<b>main.cpp</b>	1213
<b>sortingmethods.h</b>	1415

### Классы

#### Класс Goods

Класс продуктов, поставляемых на экспорт

`#include <exportGoods.h>`

#### Открытые члены

- **Goods ()**  
*Стандартный конструктор без параметров*

- **Goods** (const **Goods** &good)=default  
*Стандартный конструктор копирования*
- **Goods** (std::string name, std::string country, int quant, int price)  
*Стандартный конструктор с параметрами*
- **~Goods** ()=default  
*Стандартный деструктор*
- std::string **getName** () const  
*Get-тер для значения поля `prod_name`.*
- std::string **getCountry** () const  
*Get-тер для значения поля `export_country`.*
- int **getQuant** () const  
*Get-тер для значения поля `quant`.*
- int **getPrice** () const  
*Get-тер для значения поля `price`.*
- bool **operator<** (const **Goods** &good) const  
*Оператор сравнения <.*
- bool **operator>** (const **Goods** &good) const  
*Оператор сравнения >*
- bool **operator>=** (const **Goods** &good) const  
*Оператор сравнения >=.*
- bool **operator<=** (const **Goods** &good) const  
*Оператор сравнения <=.*
- **Goods & operator=** (const **Goods** &good)=default  
*Стандартный оператор =.*

---

## Подробное описание

Класс продуктов, поставляемых на экспорт

Класс, объекты которого будут подвергнуты различным сортировкам в соответствии с заданием. Класс имеет следующие поля:

## Параметры шаблона

<i>prod_name</i>	- Название товара
<i>export_count</i> <i>ry</i>	- Страна, в которую экспортируются товары
<i>quant</i>	- Количество товаров
<i>price</i>	- Цена за партию товара

---

## Конструктор(ы)

### **Goods::Goods ()**

Стандартный конструктор без параметров

Проставляет всем полям класса "нулевые" значения

### **Goods::Goods (const Goods & *good*)[default]**

Стандартный конструктор копирования

Все поля *good* копируются в новый объект класса *Good*

### Аргументы

<i>good</i>	- объект подлежащий копированию
-------------	---------------------------------

### **Goods::Goods (std::string *name*, std::string *country*, int *quant*, int *price*)**

Стандартный конструктор с параметрами

### Аргументы

<i>name</i>	- имя товара
<i>country</i>	- страна экспорта
<i>quant</i>	- количество товара

<i>price</i>	- цена товара
--------------	---------------

**Goods::~~Goods ()[default]**

Стандартный деструктор

---

## Методы

**std::string Goods::getCountry () const[inline]**

Get-тер для значения поля export\_country.

### Возвращает

Значение поля export\_country - страну экспорту

**std::string Goods::getName () const[inline]**

Get-тер для значения поля prod\_name.

### Возвращает

Значения поля prod\_name - название товара

**int Goods::getPrice () const[inline]**

Get-тер для значения поля price.

### Возвращает

Значения поля price - цену товара

**int Goods::getQuant () const[inline]**

Get-тер для значения поля quant.

### Возвращает

Значения поля quant - количество товара

**bool Goods::operator< (const Goods & *good*) const**

Оператор сравнения <.

Сравнивает по полям: название товара, его количество, страна экспорта Сравнение происходит по убыванию приоритета

**Аргументы**

<i>good</i>	- правый операнд
-------------	------------------

**Возвращает**

true либо false в зависимости от результата сравнения

**bool Goods::operator<= (const Goods & *good*) const**

Оператор сравнения <=.

Сравнивает по полям: название товара, его количество, страна экспорта Сравнение происходит по убыванию приоритета

**Аргументы**

<i>good</i>	- правый операнд
-------------	------------------

**Возвращает**

true либо false в зависимости от результата сравнения

**Goods & Goods::operator= (const Goods & *good*)[default]**

Стандартный оператор =.

Позволяет присваивать объекта класса друг к другу (копировать)

**bool Goods::operator> (const Goods & *good*) const**

Оператор сравнения >

Сравнивает по полям: название товара, его количество, страна экспорта Сравнение происходит по убыванию приоритета

**Аргументы**

<i>good</i>	- правый операнд
-------------	------------------

**Возвращает**

true либо false в зависимости от результата сравнения



**bool Goods::operator>= (const Goods & *good*) const**

Оператор сравнения >=.

Сравнивает по полям: название товара, его количество, страна экспорта Сравнение происходит по убыванию приоритета

#### **Аргументы**

<i>good</i>	- правый операнд
-------------	------------------

#### **Возвращает**

true либо false в зависимости от результата сравнения

---

**Объявления и описания членов классов находятся в файлах:**  
**exportGoods.h****exportGoods.cpp**

## Файлы

### Файл exportGoods.cpp

```
#include "exportGoods.h"
```

### Функции

- `std::ostream & operator<< (std::ostream &out, const Goods &good)`  
*Перегруженный оператор вставки потока <<.*

---

### Функции

`std::ostream & operator<< (std::ostream & out, const Goods & good)`

Перегруженный оператор вставки потока <<.

Позволяет эффективно выводить все поля класса в виде форматированной строки

## Файл exportGoods.h

```
#include <iomanip>
#include <iostream>
#include <string>
```

## Классы

class **Goods***Класс продуктов, поставляемых на экспорт*

## Функции

- `std::ostream & operator<< (std::ostream &out, const Goods &good)`  
*Перегруженный оператор вставки потока <<.*

---

## Функции

`std::ostream & operator<< (std::ostream & out, const Goods &  
good)`

Перегруженный оператор вставки потока <<.

Позволяет эффективно выводить все поля класса в виде  
форматированной строки

## exportGoods.h

См. документацию.

```
1 #ifndef EXPORTGOODS_H
2 #define EXPORTGOODS_H
5 #include <iomanip>
6 #include <iostream>
7 #include <string>
8
20 class Goods {
21     private:
22         std::string prod_name;
23         std::string export_country;
24         int quant, price;
25
26     public:
31         Goods();
32
38         Goods(const Goods& good) = default;
39
46         Goods(std::string name, std::string country, int quant, int price);
47
49         ~Goods() = default;
50
55         std::string getName() const { return prod_name; }
56
60         std::string getCountry() const { return export_country; }
61
66         int getQuant() const { return quant; }
67
72         int getPrice() const { return price; }
73
81         bool operator<(const Goods& good) const;
82
90         bool operator>(const Goods& good) const;
91
99         bool operator>=(const Goods& good) const;
100
108         bool operator<=(const Goods& good) const;
109
114         Goods& operator=(const Goods& good) = default;
115 };
116
121 std::ostream& operator<<(std::ostream& out, const Goods& good);
122
123 #endif // EXPORTGOODS_H
```

## Файл main.cpp

```
#include <ctime>
#include <fstream>
#include "exportGoods.h"
#include "sortingmethods.h"
```

## Макросы

- **#define N 100050**  
*Устанавливает количество строк, которые будут прочитаны из файла*
- **#define cycle 9**  
*Устанавливает, сколько измерений необходимо прозвести (число от 0 до 9)*

## Функции

- **int main ()**  
*Основная функция в программе*

---

## Макросы

**#define cycle 9**

Устанавливает, сколько измерений необходимо прозвести (число от 0 до 9)

**#define N 100050**

Устанавливает количество строк, которые будут прочитаны из файла

---

## Функции

**int main ()**

## Основная функция в программе

Здесь выполняются все необходимые по заданию операции: происходит чтение файла, состоящего из 100050 строк. Затем, проводится 9 измерений времени (с разным количеством сортируемых элементов) для разных сортировок.

## Возвращает

ноль, если программа завершилась успешно

В начале программы создаётся вектор `data`, в который считывается весь файл.

```
std::vector<Goods> data;
```

Стоит обратить внимание, что данные помещаются в вектор с помощью `emplace_back`, дабы избежать излишнего копирования

```
data.emplace_back(name, country, quant, price);
```

Для проведения измерений задаётся статически массив, в котором написано количество элементов, которые будут участвовать в сортировке на каждом конкретном замере

```
int sizes[9] = {100, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100050};
```

До начала замеров, открывается файл `datagraph.txt` на запись, в который запишутся данные для дальнейшего построения графиков некоторыми средствами визуализации данных

```
std::ofstream graph("datagraph.txt");
```

## Файл `sortingmethods.h`

`#include <vector>`

### Функции

- `template<typename T> void bubblesort (std::vector< T> *a, int const size)`  
*Функция сортировка массива "Пузырьком".*
- `template<typename T> void shakersort (std::vector< T> *a, int const size)`  
*Функция сортировка массива "Шейкер".*
- `template<typename T> void merge (std::vector< T> *a, int left, int mid, int right)`  
*Функция Слияния двух массивов*
- `template<typename T> void mergesort (std::vector< T> *a, int begin, int end)`  
*Функция сортировка массива "Слиянием".*

---

### Функции

`template<typename T> void bubblesort (std::vector< T> * a, int const size)`

Функция сортировка массива "Пузырьком".

Стандартная сортировка, просматривающая массив с конца и перемещающая наименьший элемент в начало.

#### Параметры шаблона

<i>T</i>	- определяет тип, сортируемых объектов
----------	--

#### Аргументы

<i>a</i>	- массив (вектор) значений
<i>size</i>	- размер массива

```
template<typename T> void merge (std::vector< T> * a, int left,
int mid, int right)
```

Функция Слияния двух массивов

Сливает два небольших массива (в качестве параметра передаётся один массив и место разбиения его на два) в один по некоторому правилу, которое делает сливаемый массив отсортированным.

#### Параметры шаблона

<i>T</i>	- определяет тип объектов массива
----------	-----------------------------------

#### Аргументы

<i>a</i>	- массив (вектор) значений
<i>left</i>	- левая граница массива, который нужно слить
<i>mid</i>	- место разбиения массива (вместе с <i>left</i> определяют первый массив)
<i>right</i>	- правая граница массива (вместе с <i>mid</i> определяют второй массив)

```
template<typename T> void mergesort (std::vector< T> * a, int
begin, int end)
```

Функция сортировка массива "Слиянием".

Ускоренная сортировка, делящая массива на две части много раз, которые затем сливаются в отсортированные массивы

#### Параметры шаблона

<i>T</i>	- определяет тип, сортируемых объектов
----------	--

#### Аргументы

<i>a</i>	- массив (вектор) значений массива
<i>begin</i>	- индекс начала массива для сортировки
<i>end</i>	- индекс конца массива для сортировки

```
template<typename T> void shakersort (std::vector< T> * a, int
const size)
```

Функция сортировка массива "Шейкер".

Улучшенная "пузырьковая" сортировка, просматривающая массив и с конца, и с начала по очереди.

#### Параметры шаблона

<i>T</i>	- определяет тип, сортируемых объектов
----------	--



### Аргументы

<i>a</i>	- массив (вектор) значений
<i>size</i>	- размер массива

## sortingmethods.h

См. документацию.

```
1 #ifndef SORTINGMETHODS_H
2 #define SORTINGMETHODS_H
3 #include <vector>
4
14 template <typename T>
15 void bubblesort(std::vector<T>* a, int const size) {
16     T x;
17     for (int i = 0; i < size; i++) {
18         for (int j = size - 1; j > i; --j)
19             if ((*a)[j] < (*a)[j - 1]) {
20                 x = (*a)[j - 1];
21                 (*a)[j - 1] = (*a)[j];
22                 (*a)[j] = x;
23             }
24     }
25 }
26
36 template <typename T>
37 void shakersort(std::vector<T>* a, int const size) {
38     int k = size - 1;
39     int lb, rb = size - 1;
40     T x;
41
42     do {
43         // from bottom to top passage
44         for (int i = rb; i > 0; i--) {
45             if ((*a)[i - 1] > (*a)[i]) {
46                 x = (*a)[i - 1];
47                 (*a)[i - 1] = (*a)[i];
48                 (*a)[i] = x;
49                 k = i;
50             }
51         }
52         lb = k + 1; // all elements from the start sorted
53
54         // passage from top to bottom
55         for (int j = 1; j <= rb; j++) {
56             if ((*a)[j - 1] > (*a)[j]) {
57                 x = (*a)[j - 1];
58                 (*a)[j - 1] = (*a)[j];
59                 (*a)[j] = x;
60                 k = j;
61             }
62         }
63         rb = k - 1; // all elements to the end sorted
64     } while (lb < rb);
65 }
66
79 template <typename T>
80 void merge(std::vector<T>* a, int left, int mid, int right) {
81     std::vector<T> b(right + 1 - left);
82     int i = 0;
83     int first = left, second = mid + 1; // h ; j
84     // Merges the two array's into b[] until the first one is finish
85     while ((first <= mid) && (second <= right)) {
86         if ((*a)[first] <= (*a)[second]) {
87             b[i] = (*a)[first];
88             first++;
89         } else {
90             b[i] = (*a)[second];
91             second++;
92         }
93         i++;
94     }
95     // Completes the array filling in it the missing values
96     if (first > mid) {
97         for (int k = second; k <= right; k++) {
98             b[i] = (*a)[k];
99             i++;
100         }
101     } else {
102         // if second > right
```

```

103     for (int k = first; k <= mid; k++) {
104         b[i] = (*a)[k];
105         i++;
106     }
107 }
108 for (int k = 0; k <= right - left; k++) {
109     (*a)[k + left] = b[k];
110 }
111 }
112
124 template <typename T>
125 void mergesort(std::vector<T>* a, int begin, int end) {
126     if (begin < end) {
127         int mid = (begin + end) / 2;
128         mergesort(a, begin, mid);
129         mergesort(a, mid + 1, end);
130         merge(a, begin, mid, end);
131     }
132 }
133
134 #endif // SORTINGMETHODS_H

```