Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №3
по дисциплине "Анализ Алгоритмов"

Тема Алгоритмы сортировки

Студент Сабуров С. М.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва

2021 г.

СОДЕРЖАНИЕ

Введение

Целью данной лабораторной работы является изучить и реализовать алгоритмы сортировки, оценить их трудоемкость.

Алгоритмы сортировки имеют широкое практическое применение. Сортировки используются в большом спектре задач, включая обработку коммерческих, сейсмических, космических данных. Часто сортировка является просто вспомогательной операцией для упорядочивания данных, упрощения последующих алгебраических действий над данными.

Сортировка применяется во многих областях программирования, например, базы данных или математические программы. Упорядоченные объекты содержатся в телефонных книгах, ведомостях налогов, в библиотеках, в оглавлениях, в словарях.

Во многих вычислительных системах на сортировку уходит больше половины машинного времени. Исходя из этого, можно заключить, что либо сортировка имеет много важных применений, либо ею часто пользуются без нужды, либо применяются в основном неэффективные алгоритмы сортировки.

В настоящее время, в связи с экспоненциально возросшими объемами данных, вопрос эффективной сортировки данных снова стал актуальным. В настоящее время в сети Интернет можно найти результаты производительности алгоритмов сортировки для ряда ведущих центров данных. При этом используются различные критерии оценки эффективности.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Рассмотреть и изучить сортировки пузырьком, вставками и выбором.
2. Реализовать каждую из этих сортировок.
3. Расчитать их трудоемкость.
4. Сравнить их временные характеристики экспериментально.
5. На основании проделанной работы сделать выводы.

1 Аналитическая часть

В данной главе будут рассмотрены алгоритмы сортировки.

1.1 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный (возрастание, в случае сортировки по убыванию, и наоборот), выполняется обмен элементов. Проходы по массиву повторяются $N - 1$ раз, но есть модифицированная версия, где если окажется, что обмены больше не нужны, значит проходы прекращаются. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на свое место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент массива перемещается на одну позицию к началу массива.

1.2 Сортировка вставками

Сортировка вставками — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

1.3 Сортировка выбором

Шаги алгоритмы:

1. Находится номер минимального значения в текущем списке.

2. Производится обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции).
3. Сортируется хвост списка, исключиая при этом из рассмотрения уже отсортированные элементы.

Для реализации устойчивости алгоритма необходимо в пункте 2. минимальный элемент непосредственно вставлять в первую неотсортированную позицию, не меняя порядок остальных элементов.

1.3 Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: пузырьком, вставками и выбором. Необходимо оценить трудоем- кость алгоритмов и проверить ее экспериментально.

— Входные данные — размер массива, массив .
— Выходные данные — Отсортированный массив.
— Ограничения, в рамках которых будет работать программа — размер массива должен быть целым положительным числом, элементы массива должен быть также числами(допустим вещественный тип).
— Функциональные требования — функции, представленные на листингах 2 - 4 должны сортировать массив в порядке возрастания.
— Требования к программному обеспечению — к программе предъявляется ряд требований:
    — на вход подаются размер массива (натуральное число) и сам массив, который нужно отсортировать;
    — на выходе — отсортированный массив.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов и рассчитана их трудоемкость, а также требования к программному обеспечению (далее – ПО).

Требования к вводу

1. На вход подается массив сравнимых элементов.
2. На выходе — отсортированный массив в заданном порядке;

.

2.1 Трудоемкость алгоритмов

2.1.1 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1. $+, -, /, \%, =, \neq, <, >, \leq, \geq, [], *, ++$ — трудоемкость 1.
2. Трудоемкость оператора выбора *if* условие *then A else B* рассчитывается, как:

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A & \text{если условие выполняется,} \\ f_B & \text{иначе.} \end{cases} \quad (1)$$

3. Трудоемкость цикла расчитывается, как:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}}). \quad (2)$$

4. Трудоемкость вызова функции равна 0.

2.2 Расчет трудоемкости

2.2.1 Вычисление трудоёмкости алгоритма сортировки пузырьком

Лучший случай: массив отсортирован, следовательно не произошло ни одного обмена.

Трудоемкость:

$$T(N) = 2 + \frac{N \cdot (N-1)}{2} \cdot 3 = 1.5N^2 - 1.5N + 2 \tag{3}$$

Худший случай: массив отсортирован в обратном порядке, в каждом случае происходил обмен.

Трудоемкость:

$$T(N) = 2 + \frac{N \cdot (N-1)}{2} \cdot 8 = 4N^2 - 4N + 2 \tag{4}$$

2.2.2 Вычисление трудоёмкости алгоритма сортировки вставками

Лучший случай: массив отсортирован, все внутренние циклы состоят всего из одной итерации.

Трудоемкость:

$$T(N) = 2 + 6N \tag{5}$$

Худший случай: массив отсортирован в обратном порядке. Каждый новый элемент сравнивается со всеми в отсортированной последовательности. Все внутренние циклы будут состоять из j итераций.

Трудоемкость.

$$T(N) = \frac{N \cdot (N-1)}{2} \cdot 10 + N + 2 = 5N^2 - 4N + 2 \tag{6}$$

2.2.3 Вычисление трудоёмкости алгоритма сортировки выбором

Лучший случай: массив отсортирован.

Трудоемкость:

$$T(N) = 2 + \frac{N \cdot (N-1)}{2} \cdot 8 + 4N = 4N^2 + 2 \tag{7}$$

Худший случай: массив отсортирован в обратном порядке, в каждом случае происходил обмен.

Трудоемкость:

$$T(N) = \frac{N \cdot (N-1)}{2} \cdot 5 + 5 \cdot N + 3 = 2.5N^2 + 2.5N + 3 \tag{8}$$

2.3 Описание структур данных

Был реализован класс Array, объединивший в себе алгоритмы работы с массивом и элементы массива.

Листинг 1: Описание класса Array

```
template <typename T>
class Array
{
private:
        std::vector<T> array;
public:
        Array() = default;
        Array(int N);
        Array(std::initializer_list<T> list_);
        Array(std::vector<T> list_);
        Array(const Array& Arr);
        ~Array() = default;

        void sort_insert();
        void sort_bubble();
        void sort_choice();
        void output_array();
        void fill_rand_arr();
        void reverse_arr();
};
```

2.4 Способы тестирования и классы эквивалентности

Была выбрана методика тестирования черным ящиком. Классы эквивалентности:

— Отсортированный массив.

— Массив, отсортированный в порядке убывания.

— Массив, заполненный случайными числами.

2.5 Схемы алгоритмов

На рисунках 1 - 3 изображены схемы реализации алгоритма соритровки методом пузырька, методом вставок, методом выбора.

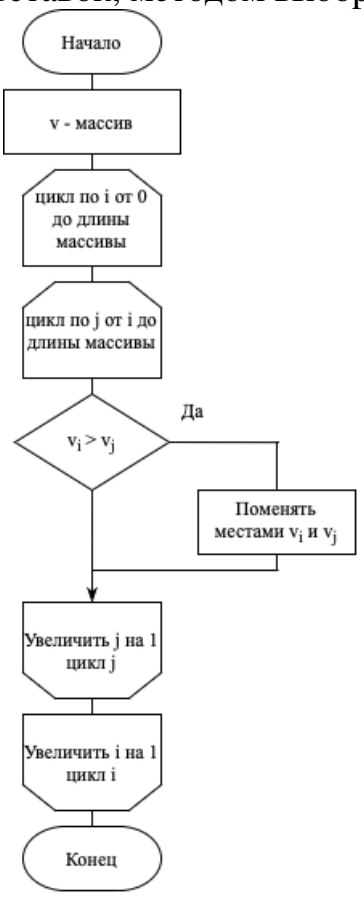

Рис. 1 — Схема алгоритма сортировки пузырьком

2.6 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов и проведена теоретическая оценка трудоемкости.

Рис. 2 — Схема алгоритма сортировки вставками

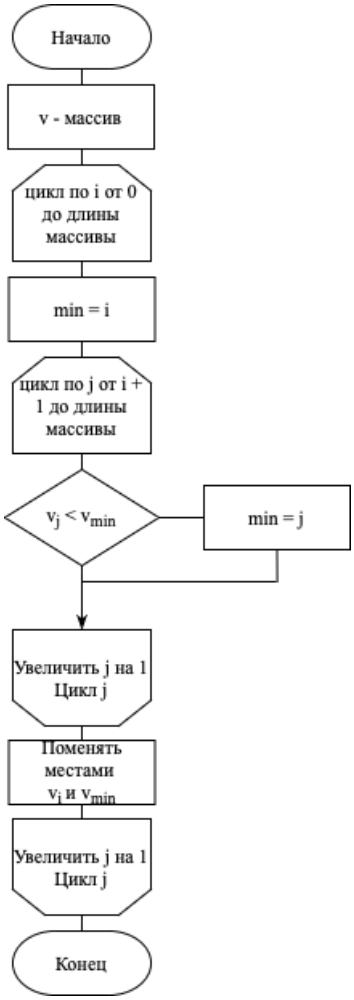

Рис. 3 — Схема алгоритма сортировки пузырьком

3 Технологическая часть

В данном разделе будет представлен выбор языка программирования и листинги реализаций алгоритмов.

3.1 Выбор языка программирования

Для реализации программ я выбрал язык программирования C++ [1]. Среда разработки Visual Studio 2019. Также данный язык был выбран потому, что в нем присутствует инструментарий для замера процессорного времени и тестирования.

3.2 Листинги реализации алгоритма

В листингах 2 - 4 представлены реализации трех алгоритмов сортировки.

Листинг 2: Функция сортировки массива методом пузырька

```
1  template <typename T>
2  void Array<T>::sort_bubble()
3  {
4
5          int l = this->array.size();
6          for (int i = 0; i < l - 1; i ++)
7          {
8                  for (int j = i; j < l; j ++)
9                  {
10                         if (this->array[i] > this->array[j])
11                         {
12                                 std::swap(this->array[i],
       this->array[j]);
13                          }
14                 }
15         }
16 }
```

Листинг 3: Функция сортировки массива методом вставок

```
1  template <typename T>
2  void Array<T>:: sort_insert()
3  {
```

```
4           for (int i = 0; i < this->array.size() - 1; i++)
5           {
6                   int key = i + 1;
7                   T temp = this->array[key];
8                   for (int j = i + 1; j > 0; j--)
9                   {
10                          if (temp < this->array[j - 1])
11                          {
12                                  this->array[j] = this->array[
    j - 1];
13                                  key = j - 1;
14                          }
15                  }
16                  this->array[key] = temp;
17          }
18  }
```

Листинг 4: Функция сортировки массива методом выбора

```
1  template <typename T>
2  void Array<T>::sort_choice()
3  {
4           for (int startIndex = 0; startIndex < this->
    array.size() - 1; ++startIndex)
5           {
6                   int smallestIndex = startIndex;
7                   for (int currentIndex = startIndex + 1;
    currentIndex < this->array.size(); ++currentIndex)
8                   {
9                           if (this->array[currentIndex] < this
    ->array[smallestIndex])
10                                  smallestIndex = currentIndex;
11                  }
12                  std::swap(this->array[startIndex], this->
    array[smallestIndex]);
13          }
14 }
```

3.3 Тестирование

Тестирование проводилось встроенной библиотекой модульного тестирования в систему сборки Cargo.

В таблице 1 приведены результаты тестирования.

Таблица 1 — Результаты функционального тестирования.

Вход	Результат	Ожидаемый результат
1, 2, 3, 4	1, 2, 3, 4	1, 2, 3, 4
4, 3, 2, 1	1, 2, 3, 4	1, 2, 3, 4
1, 3, 2, 4	1, 2, 3, 4	1, 2, 3, 4
пустой	пустой	пустой

3.3 Вывод

В данном разделе были представлены требования к программному обеспечению, выбор языка программирования, листинги реализаций алгоритмов и результаты тестирования.

4 Исследовательская часть

В данном разделе будут представлены замеры времени работы реализаций алгоритмов.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Был проведен замер времени работы каждого из алгоритмов с помощью функции GetProcessTimes. Эта функция замеряет процессорное время выполнения функции и усредняет его.

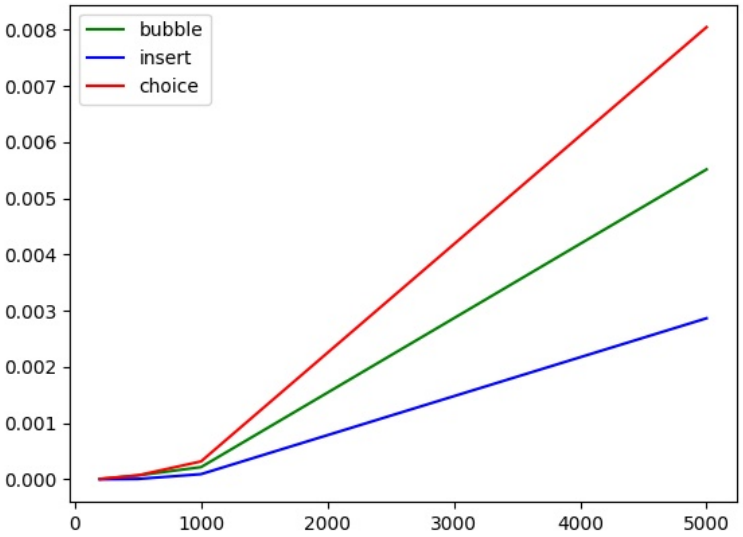

Рис. 4 — Зависимость времени выполнения алгоритмов от размера отсортированного массива. Часть 1

Таблица 2 — Результаты времени выполнения алгоритмов сортировок на отсортированном массиве (в секундах)

Размер массива	Пузырек	Вставки	Выбором
200	0.0000078125	0.0000015625	0.0000080125
500	0.0000703125	0.0000078125	0.000072125
1000	0.00021875	0.00009375	0.0003203125
5000	0.005515625	0.0028671875	0.008046875

Таблица 3 — Результаты времени выполнения алгоритмов сортировок на отсортированном в обратном порядке массиве (в секундах)

Размер массива	Пузырек	Вставки	Выбором
200	0.000015625	0.000015324	0.00000476
500	0.00021875	0.00009375	0.000109375
1000	0.000875	0.000359375	0.00034375
5000	0.01996875	0.008453125	0.0081875

4.1 Вывод

По результатам тестирования выявлено, что самым быстрым алгоритмом при использовании случайного массива оказался алгоритм сортировки методом

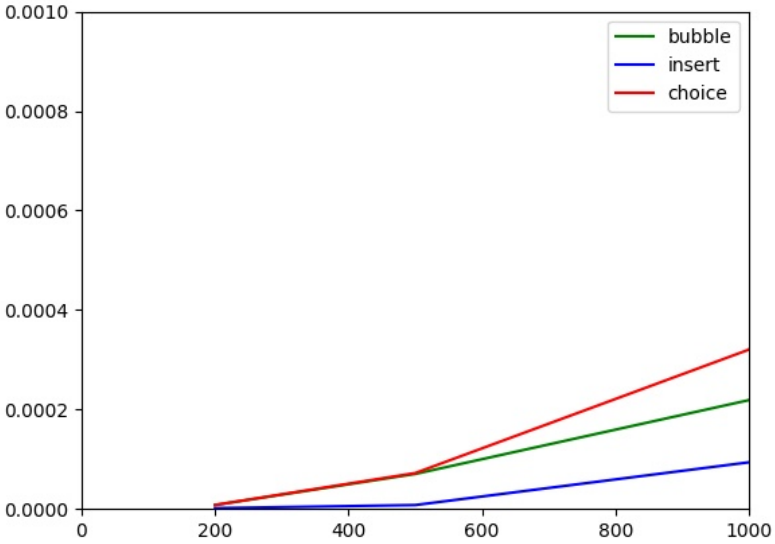

Рис. 5 — Зависимость времени выполнения алгоритмов от размера отсортированного массива. Часть 2

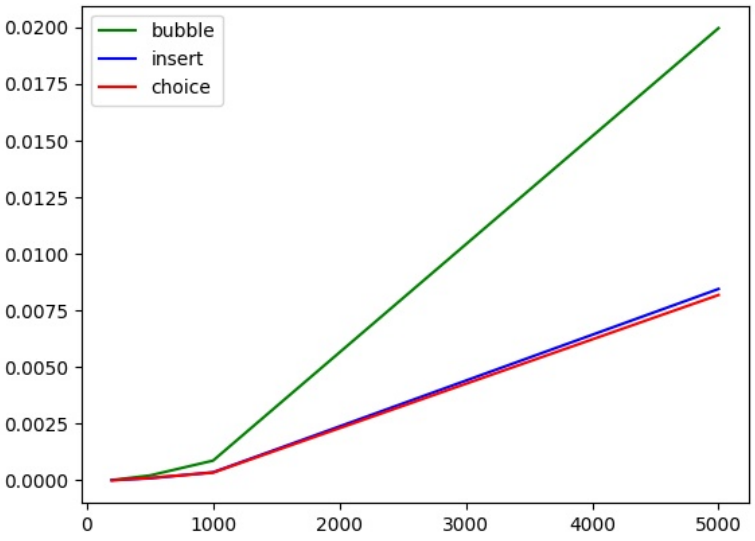

Рис. 6 — Зависимость времени выполнения алгоритмов от размера отсортированного в обратном порядке массива. Часть 1

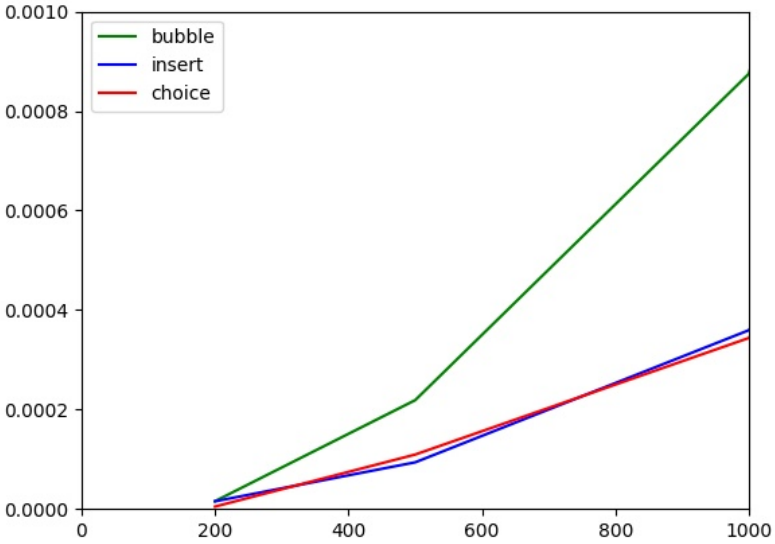

Рис. 7 — Зависимость времени выполнения алгоритмов от размера отсортированного в обратном порядке массива. Часть 2

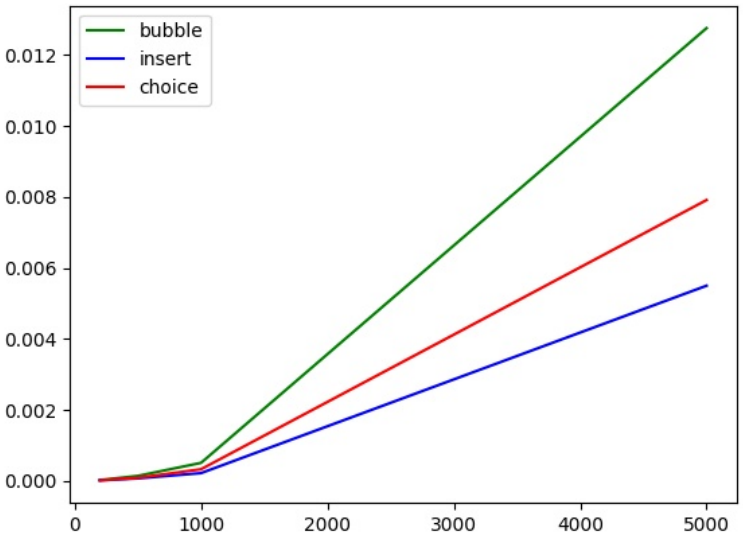

Рис. 8 — Зависимость времени выполнения алгоритмов от размера массива, заполненного случайным образом. Часть 1

Таблица 4 — Результаты времени выполнения алгоритмов сортировок на случайном массиве (в секундах)

Размер массива	Пузырек	Вставки	Выбором
200	0.0000234375	0.000015625	0.000015625
500	0.000140625	0.0000703125	0.0000859375
1000	0.0005078125	0.00021875	0.000328125
5000	0.0127578125	0.0055	0.0079140625

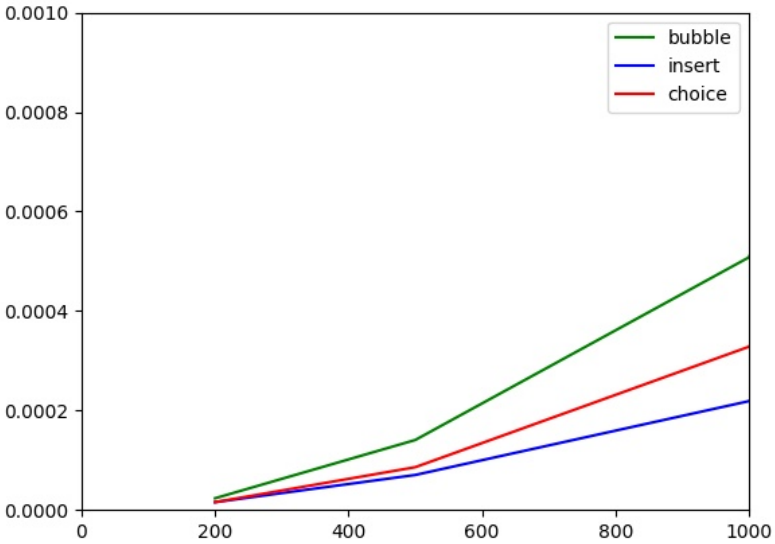

Рис. 9 — Зависимость времени выполнения алгоритмов от размера массива, заполненного случайным образом. Часть 2

вставок. Метод пузырька является самый медленным на случайном наборе данных.

Заключение

В рамках данной лабораторной работы:

— были изучены и реализованы три алгоритма сортировки: методом пузырька, вставок и выбором;

— был проведен сравнительный анализ трудоемкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;

— был проведен сравнительный анализ алгоритмов на основе экспериментальных данных;

Экспериментально были установлены различия в производительности различных алгоритмов сортировки. Для массивов длины 5000, заполненной случайными данными, метод пузырька медленнее вставок и выбора.

Список литература

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы. Построение и анализ. Издательский дом “Вильямс”, 2011. 173 - 240.
2. Б. Страуструп. Язык программирования C++ . Addison-Wesley, 2000. 142 - 145.
3. Г. Шилдт. C++. Полное руководство. СПб.: Наука и Техника, Издательский дом “Вильямс”, 2006. 621 - 670.
4. Я. Галовиц. C++17 STL. Стандартная библиотека шаблонов. Серийная библиотека программиста, 2018. 100 - 120.
5. Р. Седжвик. Фундаментальные алгоритмы C++. Diasoft, 2001. 52 - 59.