Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа №3

# Сортировки

Студент группы ИУ7-54Б,
Котов Никита

2019 г.

# Содержание

# Введение

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

Первые прототипы современных методов сортировки появились уже в XIX веке. К 1890 году для ускорения обработки данных переписи населения в США американец Герман Холлерит создал первый статистический табулятор — электромеханическую машину, предназначенную для автоматической обработки информации, записанной на перфокартах[1].

Сейчас же сортировки используются повсеместно, и именно им посвящена данная лабораторная работа.
В рамках выполнения работы необходимо решить следующие задачи:

- рассмотреть и изучить сортировки пузырьком, вставками и выбором;
- реализовать каждую из этих сортировок;
- рассчитать их трудоемкость;
- сравнить их временные характеристики экспериментально;
- на основании проделанной работы сделать выводы.

# 1 Аналитическая часть

## 1.1 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. +, -, /, %, ==, !=, <, >, <=, >=, [], ++, - – имеют трудоемкость 1
2. трудоемкость оператора выбора

   **if** условие **then**
   A
   **else**
   B

   рассчитывается, как

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

3. трудоемкость цикла рассчитывается, как $f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}})$
4. трудоемкость вызова метода равна 0
5. трудоемкость вызова функции равна 1

## 1.2 Описание алгоритмов

### 1.2.1 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный,в ыполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает - массив отсортирован. 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.

## 1.3 Сортировка вставками

Сортировка вставками - алгоритм сортировки, котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов[1].
На вход алгоритма подаётся последовательность $n$ чисел: $a_1, a_2, ..., a_n$. Сортируемые числа также называют ключами. Входная последовательность на практике представляется в виде массива с $n$ элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности $a_1^{'}, a_2^{'}, ..., a_n^{'}$, чтобы выполнялось следующее соотношение $a_1^{'} \leq a_2^{'} \leq ... \leq a_n^{'}$[2].

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[2].

## 1.4 Сортировка выбором

Шаги алгоритма:

1. находим номер минимального значения в текущем списке
2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. теперь сортируем хвост списка, исключив 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.

# 2 Конструкторская часть

## 2.1 Разработка алгоритмов

### 2.1.1 Сортировка пузырьком

На рис. 1 представлена схема сортировки пузырьком

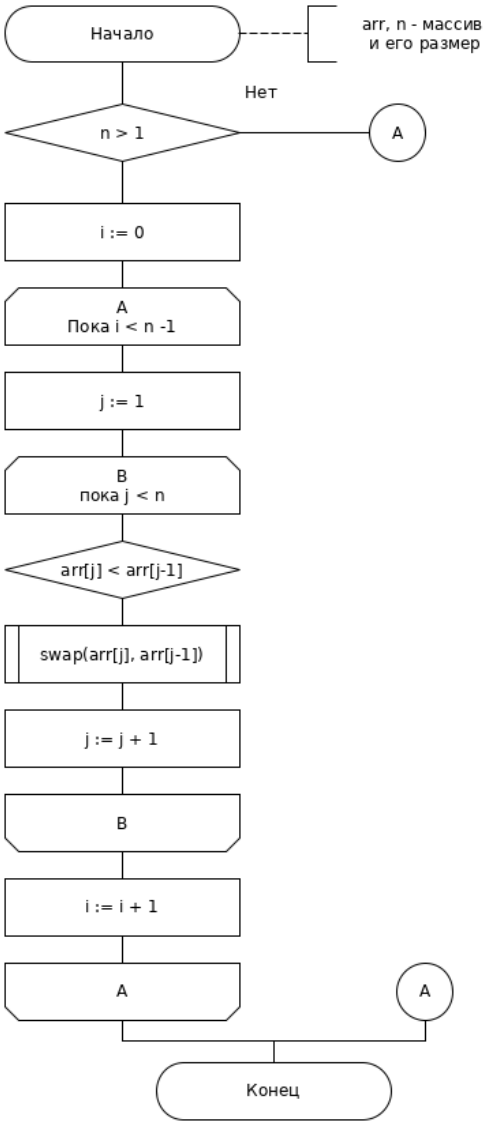


Рис. 1: Сортировка пузырьком

Лучшим случаем является ситуация, когда массив уже отсортирован, так как количество произведенных обменов будет равно 0.
Худшим же случаем является обратно отсортированный массив, так как обмен будет выполняться на каждом шаге.
Оценим трудоемкость:

- Худший случай(неотсортированный массив): $3+(n-1)(5+13(n-1)) = 13n^2-21n+11 \sim O(n^2)$
- Лудший случай(отсортированный массив): $3+(n-1)(5+6(n-1) = 6n^2-7n+1 \sim O(n^2)$

### 2.1.2 Сортировка вставками

На рис. 2 представлена схема сортировки вставками

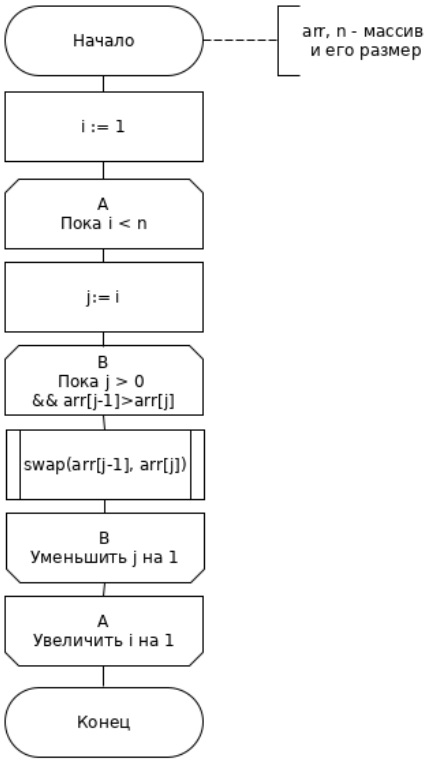


Рис. 2: Сортировка вставками

Лучшим случаем является предварительно отсортированный массив, так как тогда требуется только один проход по массиву и не производятся перестановки.
Худшим случаем является обратно отсортированный массив, т.к. при этом производится максимально возможное количество обменов. Трудоемкость алгоритма в лучшем случа $O(n)$, в худшем - $O(n^2)$[1].

### 2.1.3 Сортировка выбором

На рис. 3 представлена схема сортировки выбором

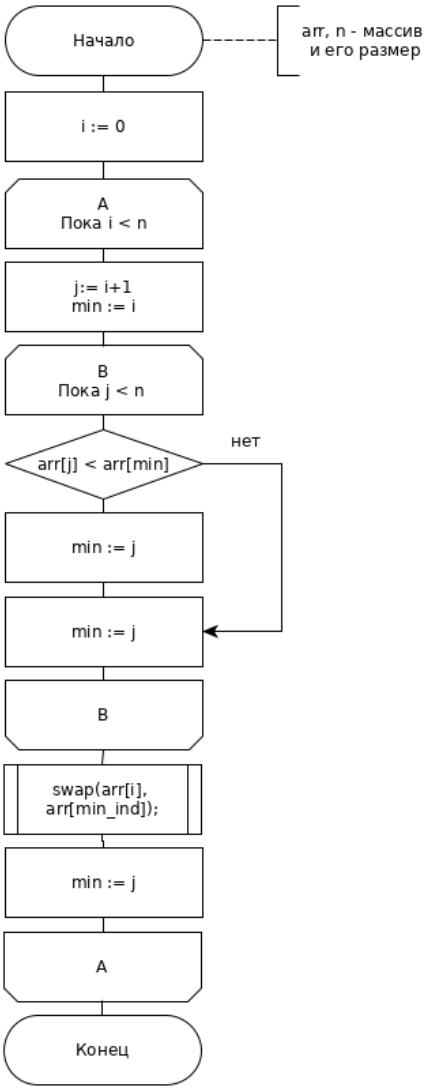


Рис. 3: Сортировка выбором

Лучшим случаем является отсортированный массив, так как при этом выполняется 1 операцию присваивания меньше на каждой итерации.
Худшим же обратно отсортированный массив, на каждой итерации выполняется на 1 операцию больше по сравнению с лучшим случаем. Трудоемкость алгоритма в обоих случаях составляет $O(n^2)$[1]

## 2.2 Выводы по конструкторскому разделу

Были разработаны схемы всех трех алгоритмов сортировки. Для каждого из них были выделены и оценены лучшие и худшие случаи

# 3 Технологическая часть

## 3.1 Средства реализации

В качестве языка программирования был выбран C++, так как он предоставляет широкие возможности для эффективной реализации алгоритмов.

## 3.2 Структура программы

sort.cpp - содержит реализации алгоритмов сортировок
test.cpp - содержит тесты для реализованных алгоритмов
main.cpp - содержит интерфейс для взаимодействия с программой

## 3.3 Листинг кода

Листинг 1: Сортировка пузырьком

```
void bubble_sort(std::vector<double> &arr) {
    if (arr.empty()) {
        return;
    }

    for (int i = 0; i < arr.size() - 1; i++) {
        for (int j = 1; j < arr.size(); j++) {
            if (arr[j] < arr[j - 1]) {
                std::swap(arr[j - 1], arr[j]);
            }
        }
    }
}
```

Листинг 2: Сортировка вставками

```
void insertion_sort(std::vector<double> &arr) {
    if (arr.empty()) {
        return;
    }

    for (size_t i = 1; i < arr.size(); i++) {
        auto cur_item = arr[i];
        int j = static_cast<int>(i - 1);
        for (; j >= 0 && arr[j] > cur_item; j--) {
            arr[j + 1] = arr[j];
        }
        arr[j + 1] = cur_item;
    }
}
```

Листинг 3: Сортировка выбором

```
void choice_sort(std::vector<double> &arr) {
    if (arr.empty()) {
        return;
    }

    for (size_t i = 0; i < arr.size() - 1; i++) {
        auto min_ind = i;
        for (size_t j = i + 1; j < arr.size(); j++) {
            if (arr[j] < arr[min_ind]) {
                min_ind = j;
            }
        }
        std::swap(arr[min_ind], arr[i]);
    }
}
```

## 3.4 Тестирование фунций

В таблице  1, таблице 2, таблице 3 приведены результаты тестирования для сортировок пузырьком, вставками и выбором соответственно.

Таблица 1: Сортировка пузырьком

| Входной массив | Результат | Ожидаемый результат |
|---|---|---|
| $[1, 2, 3, 4]$ | $[1, 2, 3, 4]$ | $[1, 2, 3, 4]$ |
| $[3, 2, 1]$ | $[1, 2, 3]$ | $[1, 2, 3]$ |
| $[5, 6, 2, 4, -2]$ | $[-2, 2, 4, 5, 6]$ | $[-2, 2, 4, 5, 6]$ |
| $[4]$ | $[4]$ | $[4]$ |
| $[]$ | $[]$ | $[]$ |

Таблица 2: Сортировка вставками

| Входной массив | Результат | Ожидаемый результат |
|---|---|---|
| $[1, 2, 3, 4]$ | $[1, 2, 3, 4]$ | $[1, 2, 3, 4]$ |
| $[3, 2, 1]$ | $[1, 2, 3]$ | $[1, 2, 3]$ |
| $[5, 6, 2, 4, -2]$ | $[-2, 2, 4, 5, 6]$ | $[-2, 2, 4, 5, 6]$ |
| $[4]$ | $[4]$ | $[4]$ |
| $[]$ | $[]$ | $[]$ |

Таблица 3: Сортировка выбором

| Входной массив | Результат | Ожидаемый результат |
|---|---|---|
| $[1, 2, 3, 4]$ | $[1, 2, 3, 4]$ | $[1, 2, 3, 4]$ |
| $[3, 2, 1]$ | $[1, 2, 3]$ | $[1, 2, 3]$ |
| $[5, 6, 2, 4, -2]$ | $[-2, 2, 4, 5, 6]$ | $[-2, 2, 4, 5, 6]$ |
| $[4]$ | $[4]$ | $[4]$ |
| $[]$ | $[]$ | $[]$ |

# 4 Экспериментальная часть

## 4.1 Тестирование времени работы функций

Для измерения времени использовалась функция clock(). Чтобы исключить случайные отклонения в измеренном времени, измерялось время работы 10 запусков функции и делилось на 10.

В таблице 4 представлены результаты замеров времени работы сортировок на отсортированных массивах, в таблице 5 на обратно отсортированных и в таблице 6. На рис. 4 и рис. 6 изображены графики, позволяющие наглядно сравнить эффективность реализованных алгоритмов сортировок.

Таблица 4: Время работы на отсортированных массивах

| Размер массива | Пузырек | Вставки | Выбор |
|---|---|---|---|
| 100 | 0.0000760 | 0.0000002 | 0.0000042 |
| 500 | 0.0001662 | 0.0000016 | 0.0000888 |
| 1000 | 0.0006588 | 0.0000017 | 0.0003322 |
| 5000 | 0.0186644 | 0.0000084 | 0.0082122 |
| 20000 | 0.2670918 | 0.0000312 | 0.1332514 |
| 50000 | 1.8965828 | 0.0000656 | 0.8430474 |

Таблица 5: Время работы на обратно отсортированных массивах

| Размер массива | Пузырек | Вставки | Выбор |
|---|---|---|---|
| 100 | 0.0000106 | 0.0000018 | 0.0000054 |
| 500 | 0.0001930 | 0.0000198 | 0.0000878 |
| 1000 | 0.0007774 | 0.0000754 | 0.0003400 |
| 5000 | 0.0190598 | 0.0019278 | 0.0085574 |
| 20000 | 0.3052872 | 0.0299092 | 0.1368812 |
| 50000 | 1.8856402 | 0.1933902 | 0.8473344 |

Таблица 6: Время работы на произвольных массивах

| Размер массива | Пузырек | Вставки | Выбор |
|---|---|---|---|
| 100 | 0.0000106 | 0.0000014 | 0.0000058 |
| 500 | 0.0001946 | 0.0000114 | 0.0000964 |
| 1000 | 0.0007532 | 0.0000446 | 0.0003844 |
| 5000 | 0.0207424 | 0.0009334 | 0.0082266 |
| 20000 | 0.3687078 | 0.0156806 | 0.1322612 |
| 50000 | 2.3855662 | 0.0933284 | 0.8455662 |

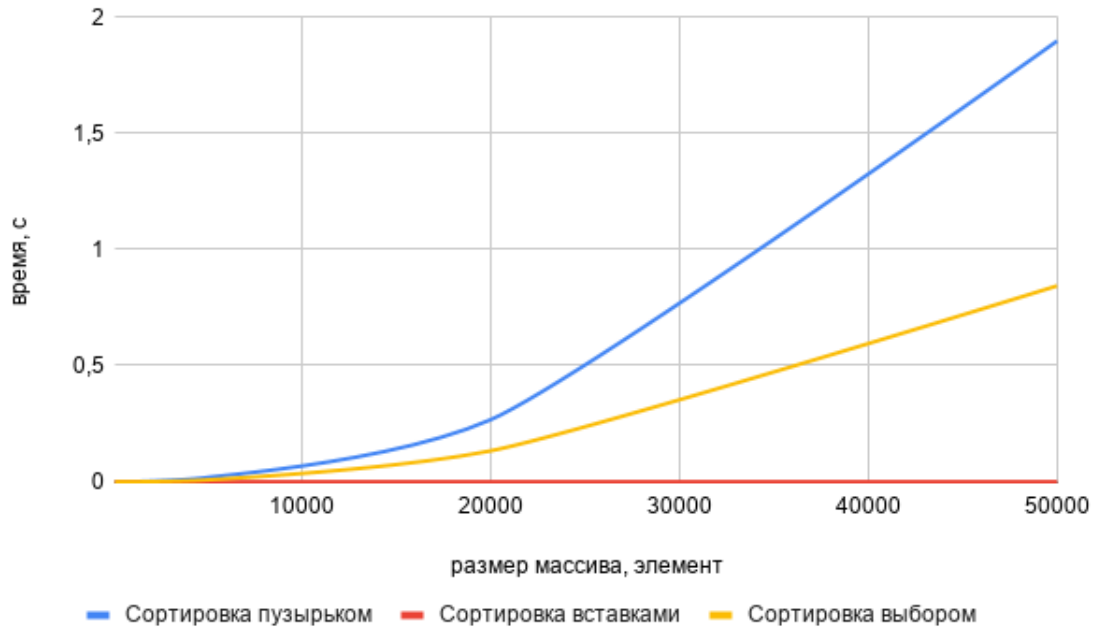


Рис. 4: Сравнение сортировок на отсортированных массивах

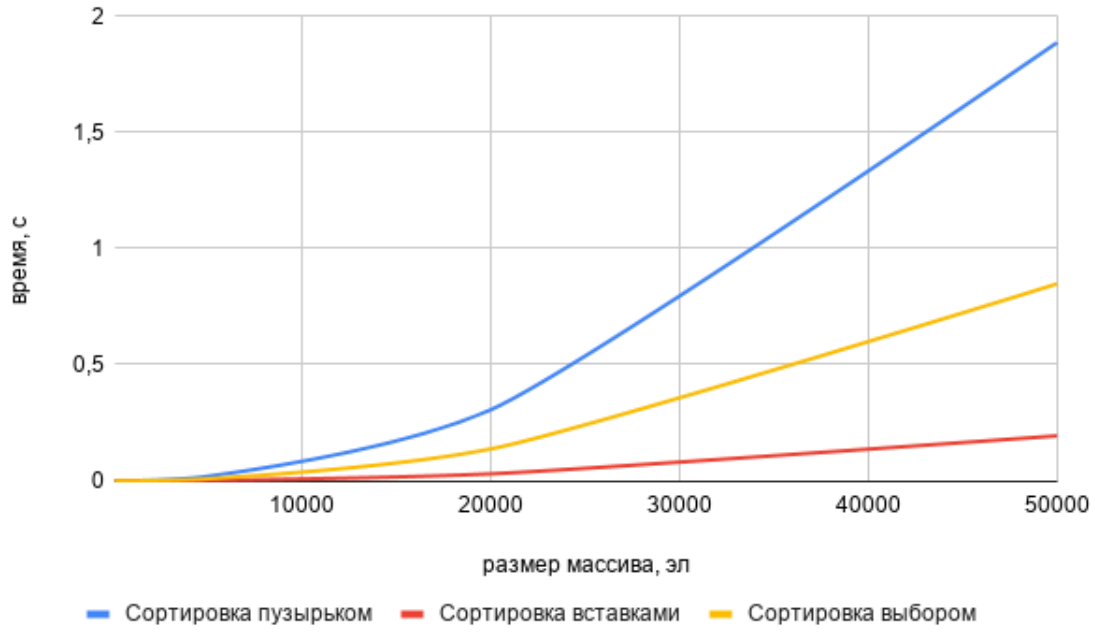


Рис. 5: Сравнение сортировок на обратно отсортированных массивах

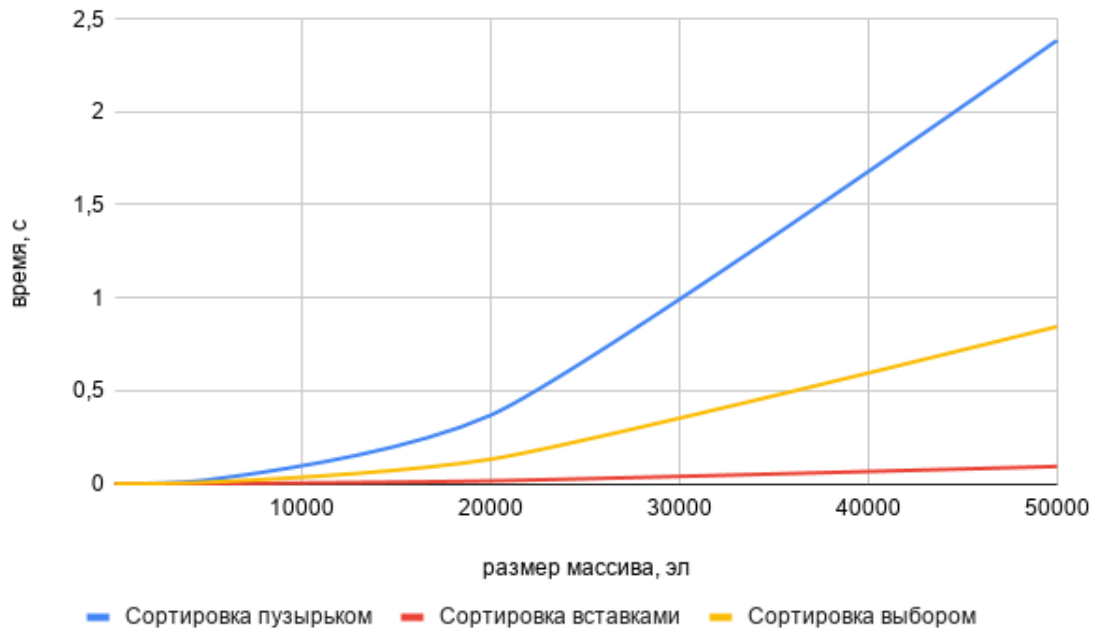


Рис. 6: Сравнение сортировок на произвольных массивах

На графиках видно, что наиболее эффективным алгоритмом является сортировка вставками, на произвольных массивах она на $\sim 95\%$ быстрее остальных. Сортировка выбором на $\sim 80\%$ быстрее пузырьково сортировки.

# Заключение

В рамках лабораторной работы были реализованы и изучены сортировки пузырьком, вставками и выбором. Была оценена их трудоемкость, произведены замеры времени работы реализованных алгоритмов и сравнена их эффективность по времени. Наиболее эффективной оказалась сортировка вставками, выигрывая по времени на $\sim 95\%$ у пузырьковой и на $\sim 80\%$ у сортировки вставками, особенно эффективна же она на отсортированных массивах, т.к. работает за линейное время.

# Список литературы

[1] Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.

[2] Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. 2.1. Сортировка вставкой // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 3-е изд. — М.: Вильямс, 2013. — С. 38—45. — ISBN 5-8459-1794-8.