**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»

# Отчет по лабораторной работе №3 по курсу «Анализ алгоритмов»

**Тема** Алгоритмы сортировки

**Студент** Кононенко С.С.

**Группа** ИУ7-53Б

**Оценка (баллы)** _____

**Преподаватели** Волкова Л.Л., Строганов Ю.В.

Москва — 2020 г.

# Оглавление

# 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.

Задачи лабораторной работы:

- 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 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 [1].

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, стр. 38–45].

## 1.3 Сортировка выбором

Шаги алгоритма:

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.

# Вывод

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

# 2 Конструкторская часть

## 2.1 Разработка алгоритмов

На рисунках 2.1, 2.2, 2.3 приведены схемы алгоритмов сортировки выбором, вставками и пузырьком соответственно.

## 2.2 Трудоёмкость алгоритмов

### 2.2.1 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1. +, -, /, %, ==, !=, <, >, <=, >=, [], *, ++, -- – трудоемкость 1.
2. Трудоемкость оператора выбора `if условие then A else B` рассчитывается, как

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.1}$$

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 функции равна 0.

### 2.2.2 Расчет трудоемкости

Оценка трудоемкости проведена по схемам 2.1, 2.2, 2.3 для алгоритмов сортировки выбором, вставками и пузырьком соответственно.

## Вычисление трудоёмкости алгоритма сортировки пузырьком

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

Трудоемкость:

$$T(N) = \frac{1+N-1}{2}(N-1) * 10 + 7N - 4 = 5N^2 + 2N - 4 \tag{2.2}$$

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

Трудоемкость:

$$T(N) = \frac{1+N-1}{2}(N-1) * 19 + 7N - 4 = 9.5N^2 - 2.5N - 4 \tag{2.3}$$

## Вычисление трудоёмкости алгоритма сортировки вставками

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

Трудоемкость:

$$T(N) = 1 + 1 + N * (1 + 1 + 1 + 4 + 1 + 2 + 1 + 1) = 12N + 2 = O(N) \tag{2.4}$$

**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:

$$\begin{aligned} T(N) = 1 + 1 + N * (1 + 1 + 1 + \sum_{1}^{N}(4 + 4 + 1 + 1) + 2 + 1) = \\ = 2 + N * (3 + N * 10 + 3) = 10N^2 + 6N + 2 = O(N^2) \end{aligned} \tag{2.5}$$

### Трудоемкость алгоритма сортировки выбором

Трудоёмкость сортировки выбором в лучшем случае $O(N^2)$ [3].
Трудоёмкость сортировки выбором в худшем случае $O(N^2)$ [3].

## Вывод

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

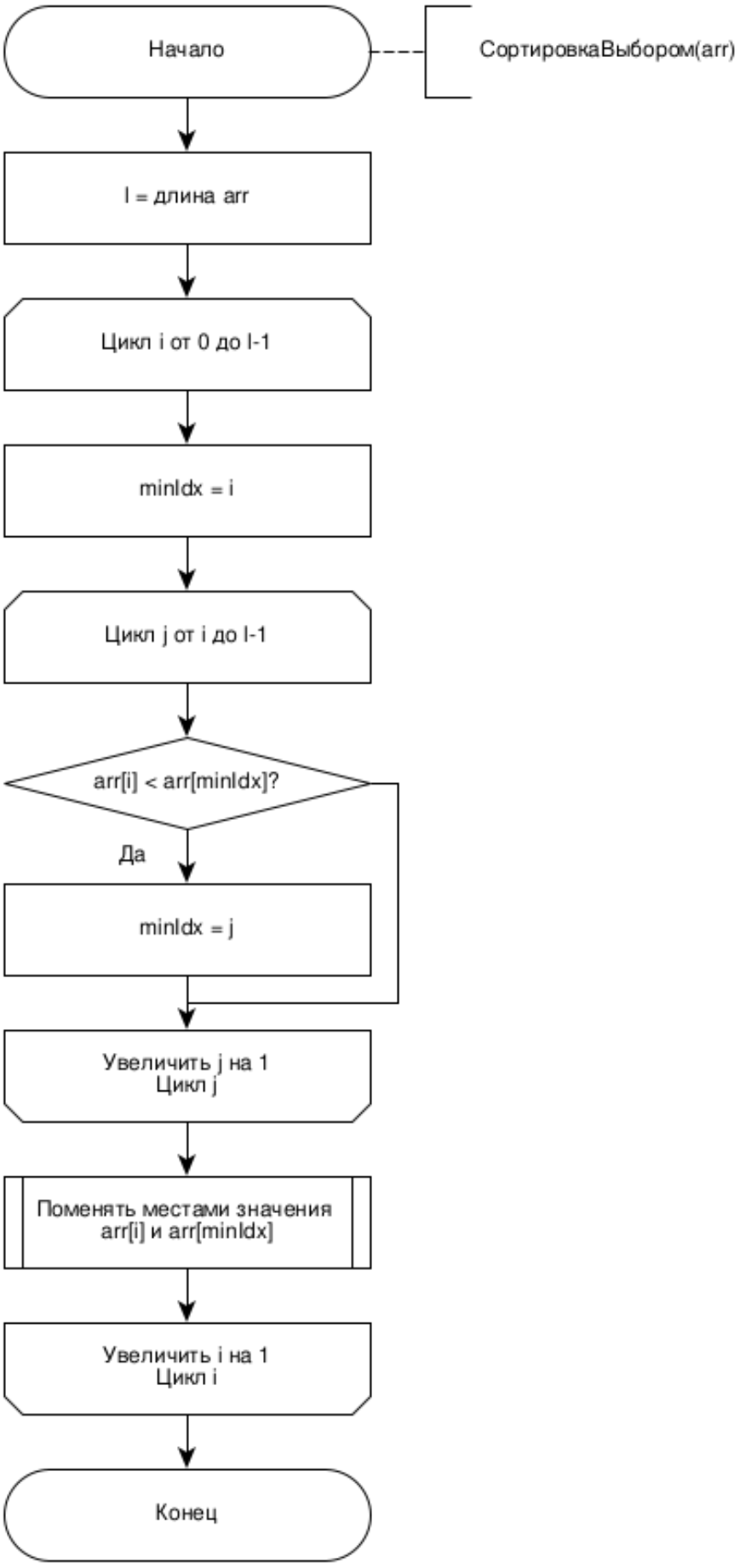


Рисунок 2.1 – Схема алгоритма сортировки выбором

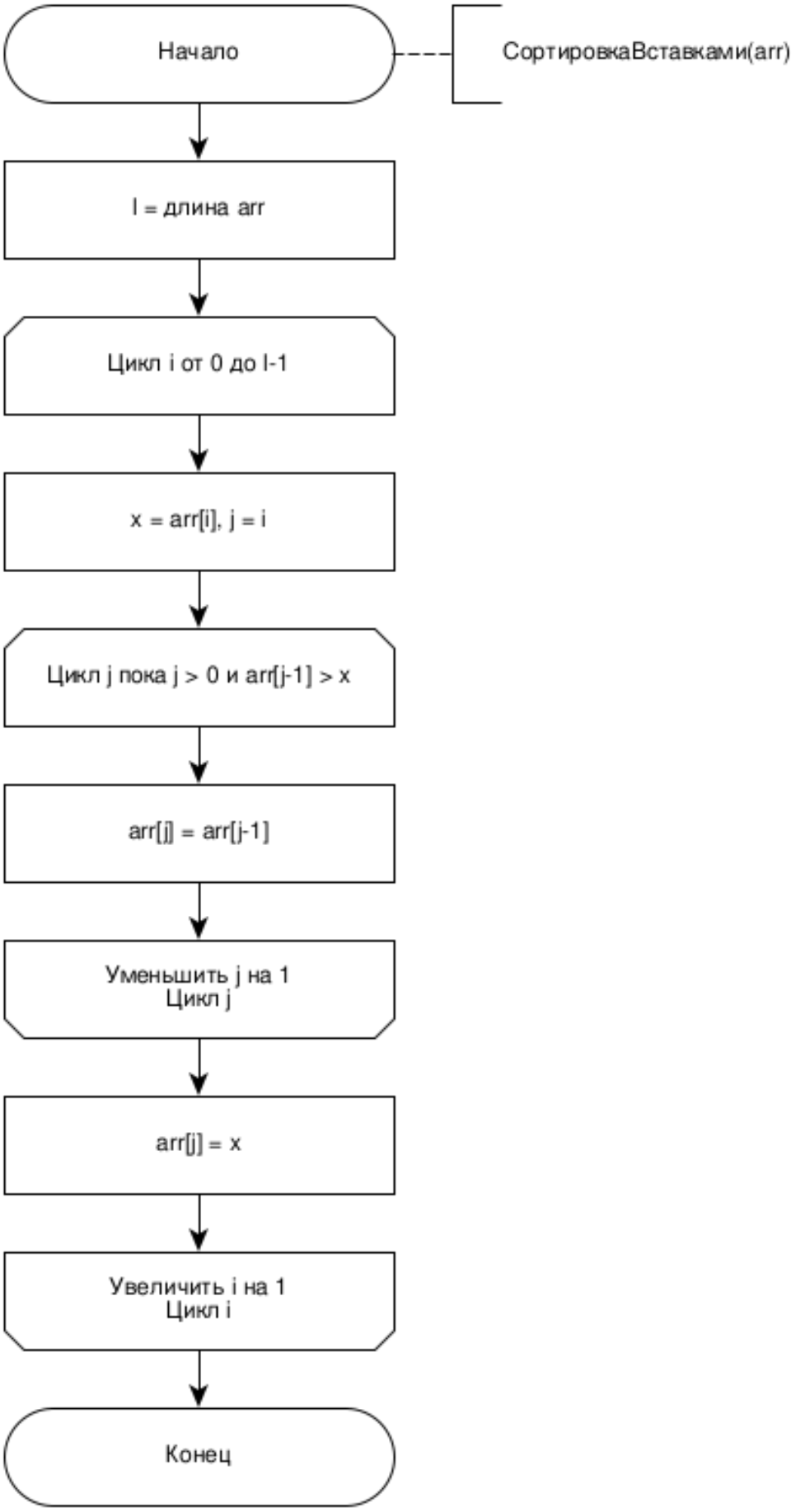


Рисунок 2.2 – Схема алгоритма сортировки вставками

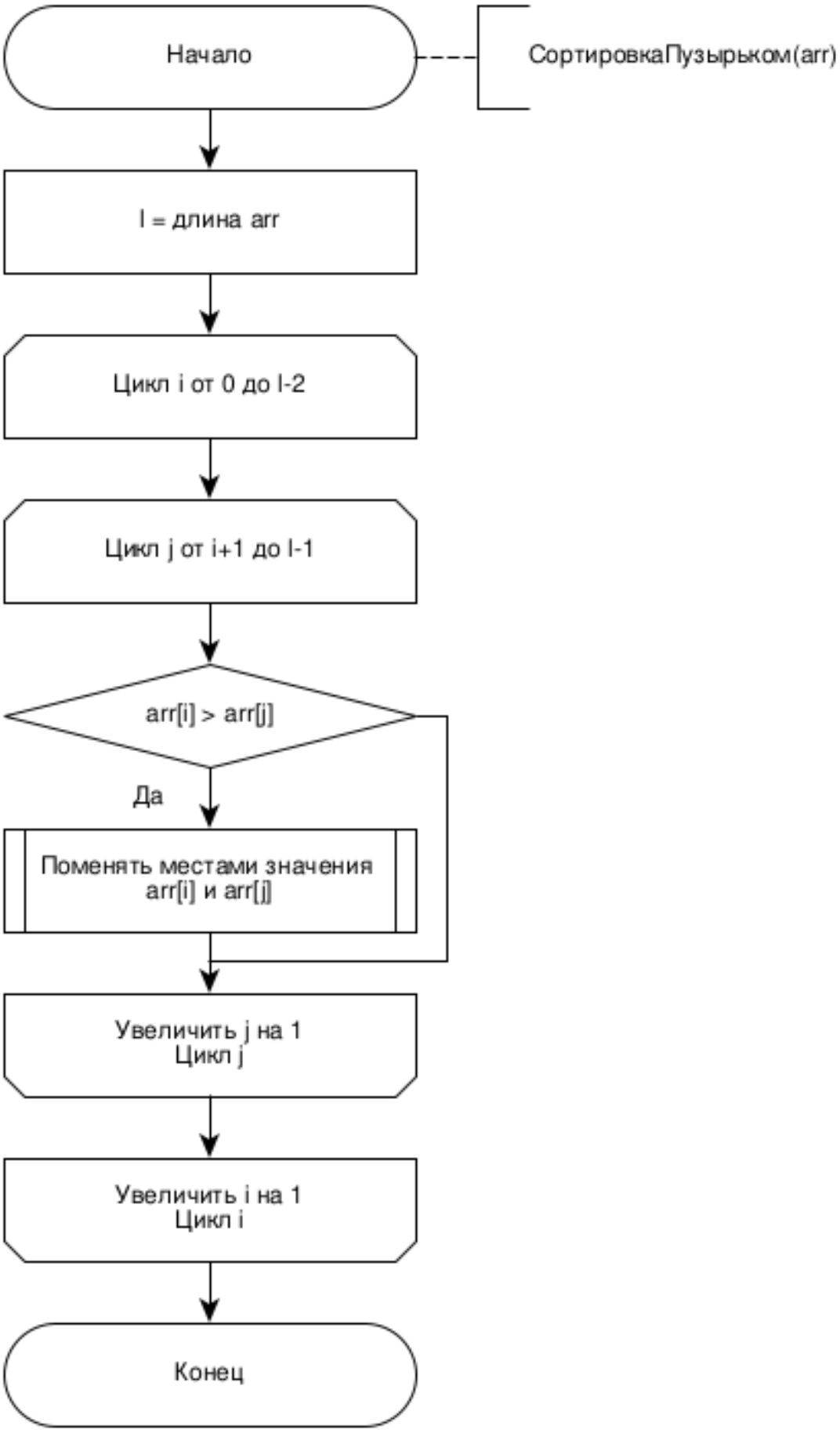


Рисунок 2.3 – Схема алгоритма сортировки пузырьком

# 3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

## 3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся размер массива (целое число) и сам массив (целые числа);
- на выходе — отсортированные при помощи сортировок выбором, вставками и пузырьком массивы.

## 3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран многопоточный язык Golang [4]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного язкыа. Помимо этого, встроенные средства языка предоставляют высокоточные средства тестирования разработанного ПО.

## 3.3 Листинг кода

В листингах 3.1, 3.2, 3.3 приведены реализации алгоритмов сортировки выбором, вставками и пузырьком соответственно.

В листинге 3.4 приведены реализации вспомогательных функций.

Листинг 3.1 – Алгоритм сортировки выбором

```
1 package sort
2
3 // Selection used to sort array using selection method.
4 func Selection(arr []int) {
5     l := len(arr)
```

```
    for i := 0; i < l; i++ {
        minIdx := i
        for j := i; j < l; j++ {
            if arr[j] < arr[minIdx] {
                minIdx = j
            }
        }
        arr[i], arr[minIdx] = arr[minIdx], arr[i]
    }
}
```

Листинг 3.2 – Алгоритм сортировки вставками

```
package sort

// Insertion used to sort array using insertion method.
func Insertion(arr []int) {
    l := len(arr)
    for i := 0; i < l; i++ {
        x := arr[i]
        j := i
        for j > 0 && arr[j-1] > x {
            arr[j] = arr[j-1]
            j--
        }
        arr[j] = x
    }
}
```

Листинг 3.3 – Алгоритм сортировки пузырьком

```
package sort

// Bubble used to sort array using bubble method.
func Bubble(arr []int) {
    l := len(arr)
    for i := 0; i < l-1; i++ {
        for j := i + 1; j < l; j++ {
            if arr[i] > arr[j] {
                arr[i], arr[j] = arr[j], arr[i]
            }
        }
    }
}
```

Листинг 3.4 – Вспомогательные функции

```
package sort

```

```go
import "fmt"

// ReadArray used to read array of n elements.
func ReadArray(n int) []int {
    arr := make([]int, n)

    for i := range arr {
        fmt.Scanf("%d", &arr[i])
    }

    return arr
}

// ReadNum used to read a word through EOL symbol.
func ReadNum() int {
    var num int
    fmt.Scanln(&num)

    return num
}
```

## 3.4 Тестирование функций

В таблице 3.1 приведены функциональные тесты для функций, реализующих алгоритмы сортировки. Все тесты пройдены успешно.

Таблица 3.1 – Тестирование функций

| Входной массив | Результат | Ожидаемый результат |
|---|---|---|
| $[1, 2, 3, 4]$ | $[1, 2, 3, 4]$ | $[1, 2, 3, 4]$ |
| $[4, 3, 2, 1]$ | $[1, 2, 3, 4]$ | $[1, 2, 3, 4]$ |
| $[8, 3, 1, -3, -2]$ | $[-3, -2, 1, 3, 8]$ | $[-3, -2, 1, 3, 8]$ |
| $[1]$ | $[1]$ | $[1]$ |
| пустой массив | пустой массив | пустой массив |

# Вывод

Были разработаны и протестированы реализации алгоритмов: сортировки выбором, вставками и пузырьком.

# 4 Исследовательская часть

## 4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

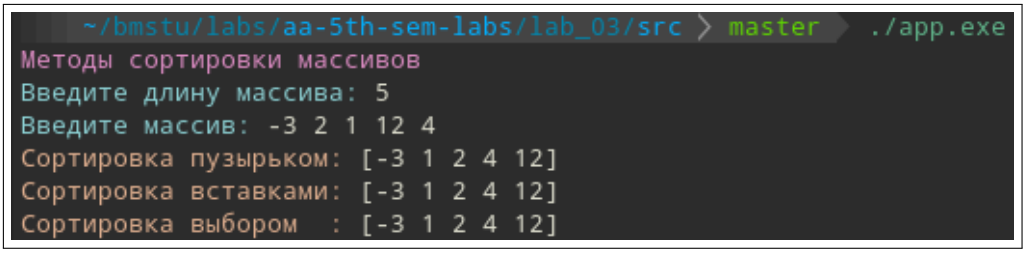


Рисунок 4.1 – Демонстрация работы алгоритмов сортировки

## 4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Manjaro [5] Linux [6] 20.1 64-битная.
- Память: 16 ГБ.
- Процессор: AMD Ryzen™ 7 3700U [7] ЦПУ @ 2.30ГГц

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования.

## 4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи написания «бенчмарков» [8], предоставялемых встроенными в Go средствами. Для такого рода тестирования не нужно самостоятельно указывать количество повторов. В библиотеке для тестирования существует константа $N$, которая динамически

варьируется в зависимости от того, был ли получен стабильный результат или нет.

В листинге 4.1 пример реализации бенчмарка.

Листинг 4.1 – Реализация бенчмарка

```
package sort

import (
    "math/rand"
    "testing"
)

// Selection sort benchmarks.

func BenchmarkSelectionSorted10(b *testing.B) {
    N := 10
    arr := make([]int, N)
    narr := make([]int, N)
    for i := 0; i < N; i++ {
        arr[i] = i
    }

    for i := 0; i < b.N; i++ {
        copy(narr, arr)
        Selection(narr)
    }
}
```

Результаты замеров приведены в таблицах 4.1, 4.2 и 4.3. На рисунках 4.2, 4.3 и 4.4 приведены графики, иллюстрирующие зависимость времени работы алгоритмов сортировки от размеров массивов на отсортированных, обратно отсортированных и случайных данных.

Таблица 4.1 – Время работы реализации алгоритмов сортировки на отсортированных данных

| **Размер** | **Время сортировки, нс** | | |
|---|---|---|---|
| | **Пузырьком** | **Вставками** | **Выбором** |
| 10 | 62.3 | 21 | 126 |
| 100 | 4466 | 168 | 6675 |
| 1000 | 352419 | 1606 | 568408 |

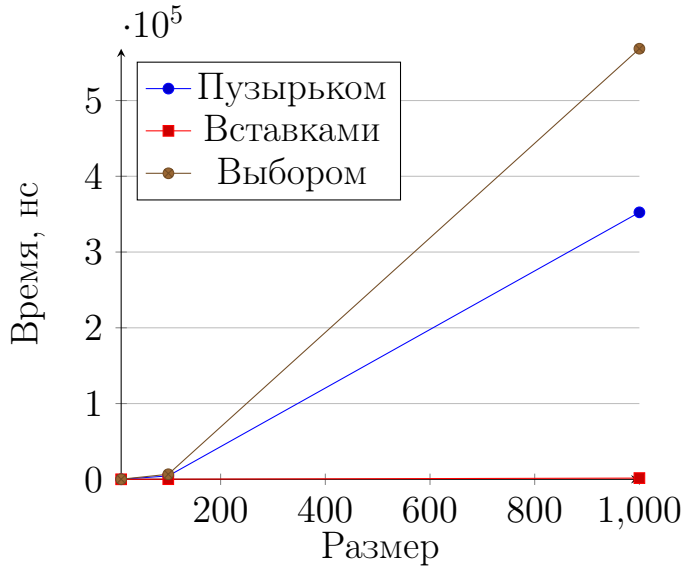


Рисунок 4.2 – Зависимость времени работы реализации алгоритма сортировки от размера отсортированного массива

Таблица 4.2 – Время работы реализации алгоритмов сортировки на обратно отсортированных данных

| **Размер** | **Время сортировки, нс** | | |
|---|---|---|---|
| | **Пузырьком** | **Вставками** | **Выбором** |
| 10 | 64.9 | 80 | 100 |
| 100 | 5480 | 6106 | 7822 |
| 1000 | 474522 | 502281 | 622797 |

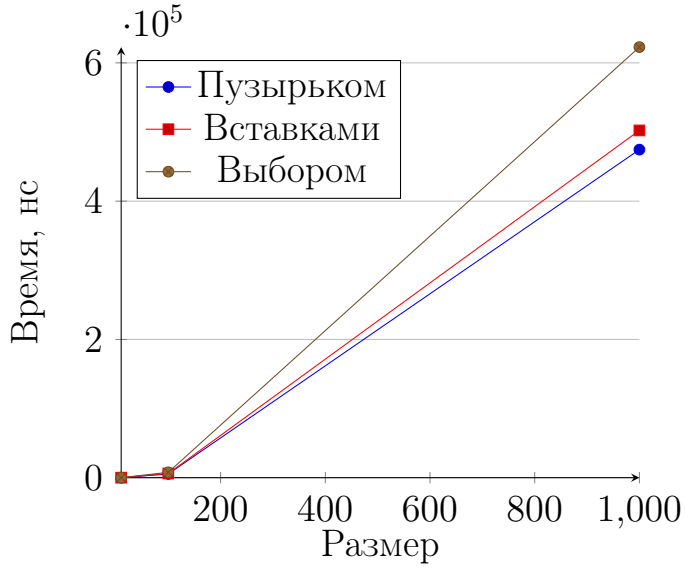


Рисунок 4.3 – Зависимость времени работы реализации алгоритма сортировки от размера обратно отсортированного массива

Таблица 4.3 – Время работы реализации алгоритмов сортировки на случайных данных

| Размер | Время сортировки, нс | | |
|---|---|---|---|
| | **Пузырьком** | **Вставками** | **Выбором** |
| 10 | 69.3 | 32.2 | 106 |
| 100 | 9540 | 2241 | 9413 |
| 1000 | 1233754 | 131675 | 614626 |

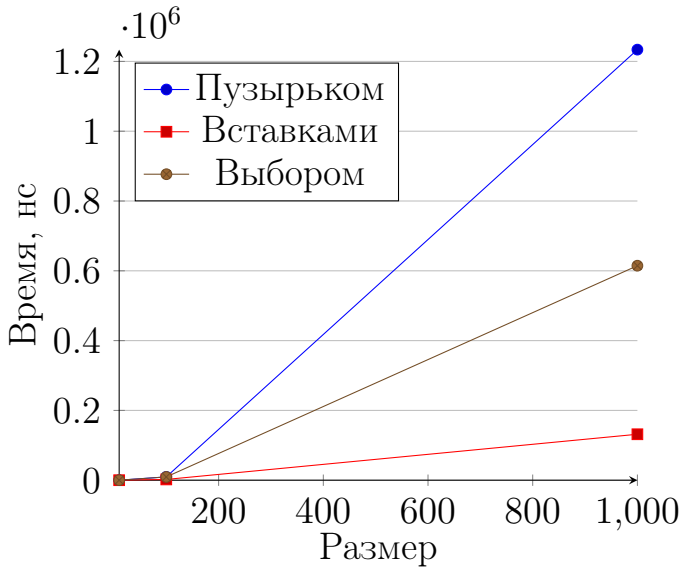


Рисунок 4.4 – Зависимость времени работы реализации алгоритма сортировки от размера случайного массива

# Вывод

Алгоритм сортировки вставками работает лучше сортировок выбором и пузырьком на случайных и уже отсортированных числах, практический интерес, конечно, представляет лишь случай сортировки случайных чисел, на котором сортировка вставками массива из 1000 элементов в 10 раз быстрее сортировки пузырьком и в 5 раз быстрее сортировки выбором. Таким образом, алгоритм сортировки вставками преуспевает над алгоритмами сортировки пузырьком и выбором.

# Заключение

В ходе выполнения работы была достигнута цель выполнены все поставленные 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.

Экспериментально были установлены различия в производительности различных алгоритмов сортировки. Для массивов длины 1000, заполненных случайным порядком, алгоритм сортировки вставками в 10 раз быстрее сортировки пузырьком и в 5 раз быстрее сортировки выбором.

Также было отмечено, что сортировка выбором работает всегда за примерно одинаковое время, поэтому в худшем случае обгоняет по времени алгоритмы сортировки выбором и пузырьком.

# Литература


[1] Кнут Д. Сортировка и поиск. – М.: Вильямс, 2000. Т. 3 из *Искусство программирования.* с. 834.

[2] Кормен Т. Лейзерсон Ч. Ривест Р. Штайн К. Алгоритмы: построение и анализ. – М.: Вильямс, 2013. с. 1296.

[3] Сортировка выбором – Википедия [Электронный ресурс]. Режим доступа: `https://ru.wikipedia.org/wiki/\T2A\CYRS\T2A\cyro\T2A\cyrr\T2A\cyrt\T2A\cyri\T2A\cyrr\T2A\cyro\T2A\cyrv\T2A\cyrk\T2A\cyra_\T2A\cyrv\T2A\cyrery\T2A\cyrb\T2A\cyro\T2A\cyrr\T2A\cyro\T2A\cyrm` (дата обращения: 14.09.2020).

[4] The Go Programming Language [Электронный ресурс]. Режим доступа: `https://golang.org/` (дата обращения: 11.09.2020).

[5] Manjaro – enjoy the simplicity [Электронный ресурс]. Режим доступа: `https://manjaro.org/` (дата обращения: 14.09.2020).

[6] Linux – Википедия [Электронный ресурс]. Режим доступа: `https://ru.wikipedia.org/wiki/Linux` (дата обращения: 14.09.2020).

[7] Мобильный процессор AMD Ryzen™ 7 3700U с графикой Radeon™ RX Vega 10 [Электронный ресурс]. Режим доступа: `https://www.amd.com/ru/products/apu/amd-ryzen-7-3700u` (дата обращения: 14.09.2020).

[8] testing – The Go Programming Language [Электронный ресурс]. Режим доступа: `https://golang.org/pkg/testing/` (дата обращения: 12.09.2020).