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”

Студент Миронов Григорий Александрович

Группа ИУ7-53Б

Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7

Тема Сортировки

Студент	_____ (подпись, дата)	Миронов Г. А. (фамилия, и.о.)
Преподаватель	_____ (подпись, дата)	Волкова Л. Л. (фамилия, и.о.)

Москва

2021 г.

Оглавление

Введение

В данной лабораторной работе будет рассмотрены различные методы сортировок.

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.

В некоторых 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.

Целью данной работы является реализация и изучение следующих алгоритмов:

- сортировка пузырьком;
- сортировка вставками;
- сортировка выбором.

Для достижения данной цели необходимо решить следующие задачи:

- рассмотреть и изучить сортировки пузырьком, вставками и выбором;
- реализовать каждую из этих сортировок;
- теоретически оцениьь их трудоемкость;
- сравнить их временные характеристики экспериментально.

1 Аналитическая часть

1.1 Описание алгоритмов

Сортировка пузырьком

Алгоритм пузырьковой сортировки совершает несколько проходов по списку. При каждом проходе происходит сравнение соседних элементов. Если порядок соседних элементов неправильный, то они меняются местами. Каждый проход начинается с начала списка.

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].

Сортировка выбором

Алгоритм сортировки выбором совершает несколько проходов по списку. При каждом проходе выбирается минимальный из еще неотсортированных элементов и помешаетяс обменивается с первым элементом неотсортированной области. В следующем проходе рассмотренный элемент не участвует, сортируется только оставшийся хвост.

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

1.2 Вывод

Пузырьковая сортировка сравнивает элементы попарно, переставляя между собой элементы тех пар, порядок в которых нарушен. Сортировка вставками, сортирует список, вставляя очередной элемент в нужное место уже отсортированного списка. Быстрая сортировка определяет опорный элемент и далее переставляет элементы, относительно выбранного элемента.

Были рассмотрены основополагающие материалы, которые в дальнейшем потребуются при реализации алгоритмов сортировки.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы вышеизложенных алгоритмов.

2.1 Схемы

На рисунке 2.1 представлена схема алгоритма сортировки пузырьком.

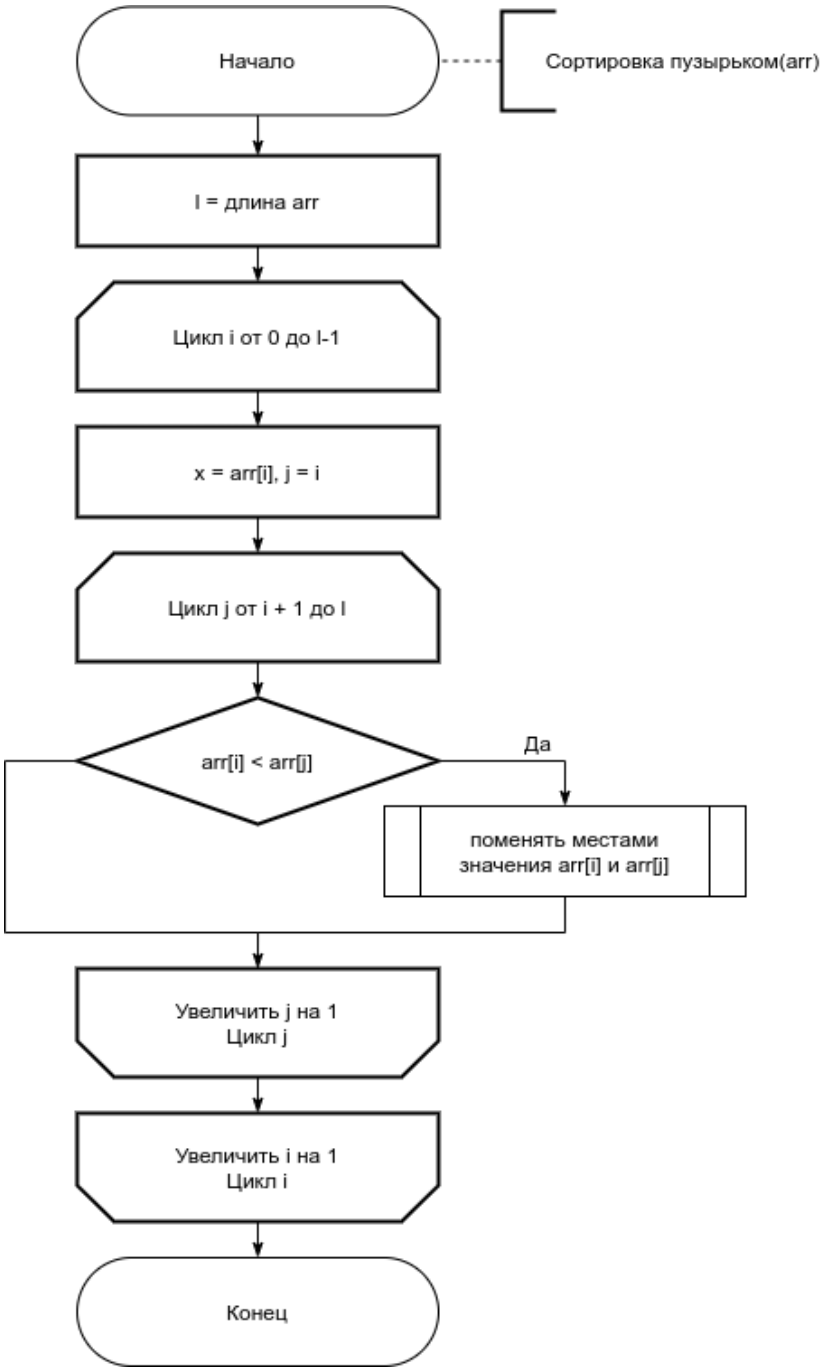

Рисунок 2.1 – Схема алгоритма сортировки пузырьком

На рисунке 2.2 представлена схема алгоритма сортировки вставками.

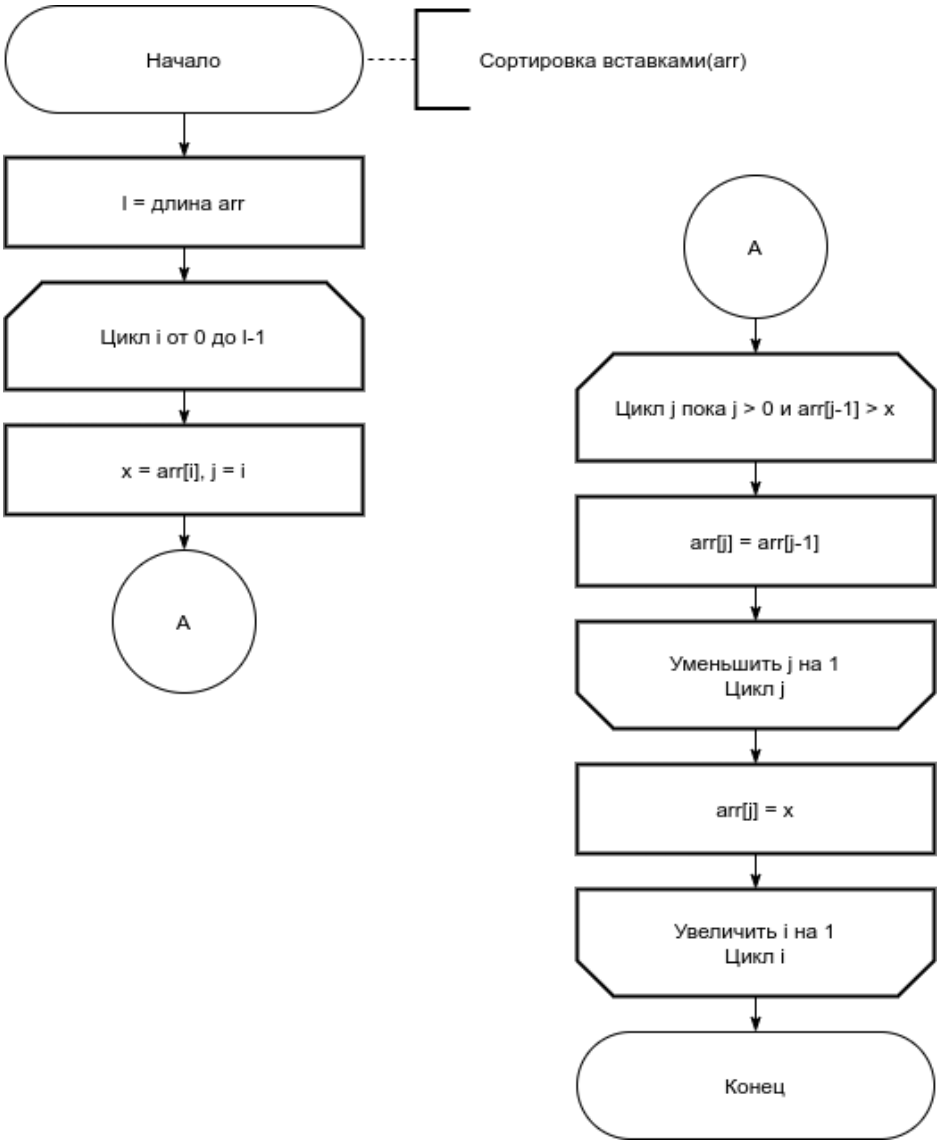

Рисунок 2.2 – Схема алгоритма сортировки вставками

На рисунке 2.3 представлена схема алгоритма сортировки выбором.

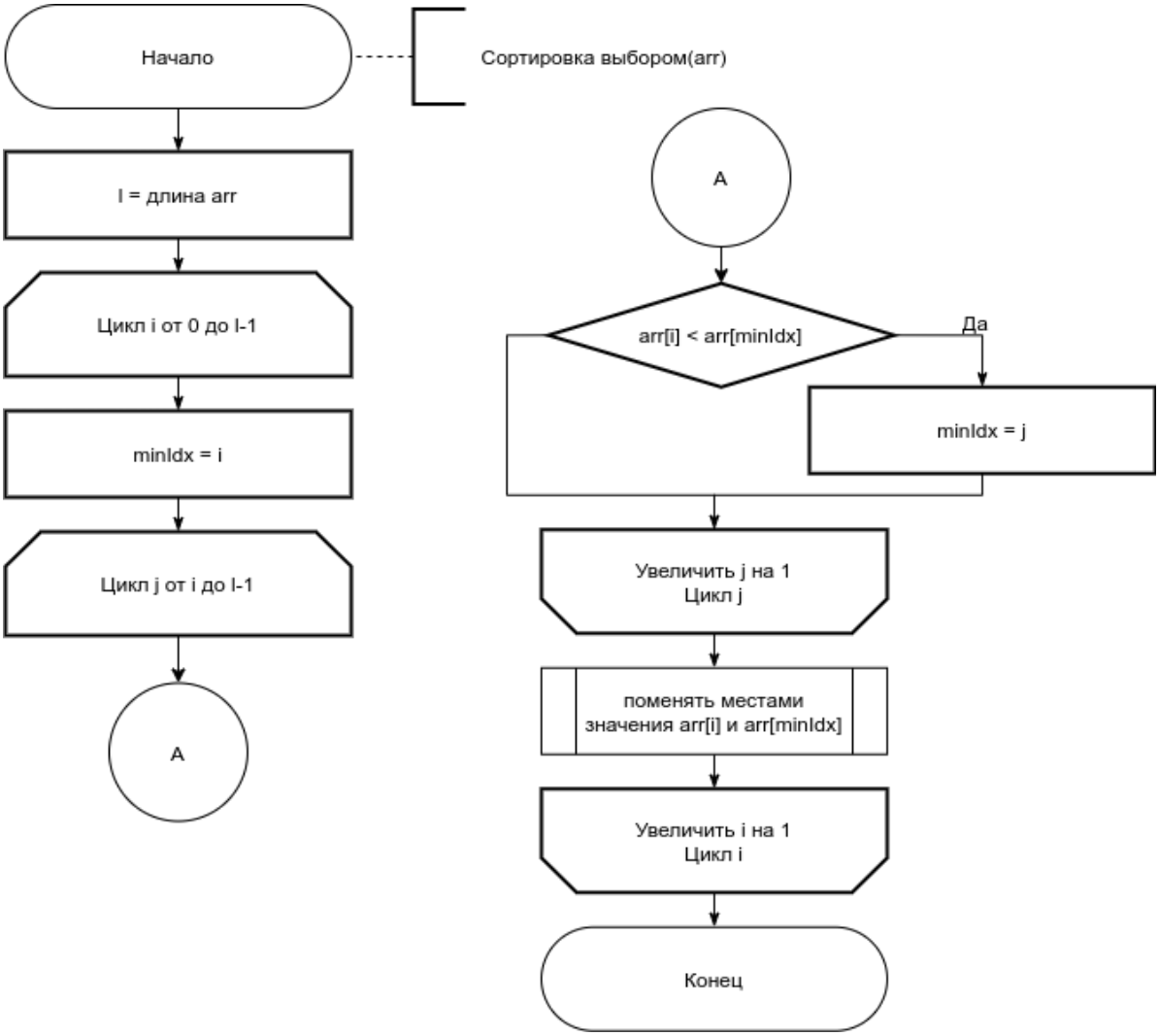

Рисунок 2.3 – Схема алгоритма сортировки выбором

2.2 Модель вычислений

Для последующего вычисления трудоемкости введём модель вычислений:

- операции из списка (2.1) имеют трудоемкость 1;

$$+, -, /, \%, ==, !=, <, >, <=, >=, [], ++, --\tag{2.1}$$

- трудоемкость оператора выбора if условие then A else B рассчитывается, как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases}\tag{2.2}$$

- трудоемкость цикла рассчитывается, как (2.3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}})\tag{2.3}$$

- трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

Пусть размер массивов во всех вычислениях обозначается как N.

2.3.1 Алгоритм сортировки пузырьком

Трудоёмкость алгоритма сортировки пузырьком состоит из:

- трудоёмкость сравнения и инкремента внешнего цикла $i \in [1..N)$ (2.4):

$$f_i = 2 + 2(N-1)\tag{2.4}$$

- суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N-1]$ (2.5):

$$f_j = 3(N-1) + \frac{N \cdot (N-1)}{2} \cdot (3 + f_{if})\tag{2.5}$$

- трудоёмкость условия во внутреннем цикле (2.6):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 9, & \text{в худшем случае} \end{cases}\tag{2.6}$$

Трудоёмкость в **лучшем** случае (2.7):

$$f_{best} = \frac{7}{2}N^2 + \frac{3}{2}N - 3 \approx \frac{7}{2}N^2 = O(N^2) \tag{2.7}$$

Трудоёмкость в **худшем** случае (2.8):

$$f_{worst} = 8N^2 - 8N - 3 \approx 8N^2 = O(N^2) \tag{2.8}$$

2.3.2 Алгоритм сортировки вставками

Трудоёмкость алгоритма сортировки вставками состоит из:

- трудоёмкость сравнения и инкремента внешнего цикла $i \in [1..N)$ (2.9):

$$f_i = 2 + 2(N - 1) \tag{2.9}$$

- суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N - 1]$ (2.10):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 3(N-1) + \frac{N\cdot(N-1)}{2} \cdot (3 + f_{if}), & \text{в худшем случае} \end{cases} \tag{2.10}$$

- трудоёмкость условия во внутреннем цикле (2.11):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 9, & \text{в худшем случае} \end{cases} \tag{2.11}$$

Трудоёмкость в **лучшем** случае (2.12):

$$f_{best} = 13N - 10 \approx 13N = O(N) \tag{2.12}$$

Трудоёмкость в **худшем** случае (2.13):

$$f_{worst} = 4.5N^2 + 10N - 13 \approx 4N^2 = O(N^2) \tag{2.13}$$

2.3.3 Алгоритм сортировки выбором

Трудоемкость сортировки выбором в лучшем случае: (N^2) [1].
Трудоемкость сортировки выбором в худшем случае: (N^2) [1].

2.4 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы (рисунки 2.1 – 2.3) трех алгоритмов сортировки. Оценены их трудоемкости в лучшем и худшем случаях. Все алгоритмы в худшем случае обладают квадратичной сложностью. А в лучшем случае меньше всего сложность у алгоритма сортировки вставками.

3 Технологическая часть

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

3.1 Выбор средств реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык Golang [3]. Данный выбор обусловлен тем, что я имею некоторый опыт разработки на нем, а так же наличием у языка встроенных высокоточных средств тестирования и анализа разработанного ПО.

3.2 Требования к программному обеспечению

Входными данными являются:

- размерность массива n;
- n элементов массива.

На выходе получается отсортированный массив.

3.3 Сведения о модулях программы

Данная программа разбита на следующие модули:

- `main.go` – Файл, содержащий точку входа в программу. В нем происходит общение с пользователем и вызов алгоритмов.
- `bubble.go` – Файл содержит реализацию сортировки пузырьком.
- `insertion.go` – Файл содержит реализацию сортировки вставками.
- `selection.go` – Файл содержит реализацию сортировки выбором.
- `utils.go` – Файл содержит различные функции для вычислений.

В листингах 3.3–3.5 представлены исходные коды разобранных ранее алгоритмов.

Листинг 3.1 – Основной файл программы main

```go
package main

import (
    "fmt"
    "github.com/logrusorgru/aurora"
    labsort "lab_03/sort"
)

func main() {
    fmt.Println(aurora.Magenta("Array sorting methods"))

    fmt.Print(aurora.Cyan("Input array size: "))
    n := labsort.ReadNum()
    fmt.Print(aurora.Cyan("Input array: "))
    arr := labsort.ReadArray(n)
    barr := make([]int, n)
    iarr := make([]int, n)
    sarr := make([]int, n)

    fmt.Print(aurora.Yellow("Bubble sort: "))
    copy(barr, arr)
    labsort.Bubble(barr)
    fmt.Println(barr)

    fmt.Print(aurora.Yellow("Insertion sort: "))
    copy(iarr, arr)
    labsort.Insertion(iarr)
    fmt.Println(iarr)

    fmt.Print(aurora.Yellow("Selection sort: "))
    copy(sarr, arr)
    labsort.Selection(sarr)
    fmt.Println(sarr)
}
```

Листинг 3.2 – Различные функции для вычислений

```
package sort

import "fmt"

func ReadArray(n int) []int {
    arr := make([]int, n)
    for i := range arr {
        fmt.Scanf("%d", &arr[i])
    }
    return arr
}

func ReadNum() int {
    var num int
    fmt.Scanln(&num)
    return num
}
```

Листинг 3.3 – Сортировка пузырьком

```
func Bubble(arr []int) {
    for i := len(arr); i > 0; i-- {
        for j := 1; j < i; j++ {
            if arr[j-1] > arr[j] {
                intermediate := arr[j]
                arr[j] = arr[j-1]
                arr[j-1] = intermediate
            }
        }
    }
}
```

Листинг 3.4 – Сортировка вставками

```
func Insertion(arr []int) {
    for i := 0; i < len(arr); i++ {
        x, j := arr[i], i
        for j > 0 && arr[j-1] > x {
            arr[j] = arr[j-1]
            j--
        }
        arr[j] = x
    }
}
```

Листинг 3.5 – Сортировка выбором

```
func Selection(arr []int) {
    for i := 0; i < len(arr); i++ {
        minIdx := i
        for j := i; j < len(arr); j++ {
            if arr[j] < arr[minIdx] {
                minIdx = j
            }
        }
        arr[i], arr[minIdx] = arr[minIdx], arr[i]
    }
}
```

3.4 Вывод

Были реализованы и протестированы спроектированные алгоритмы: вычисления расстояния Левенштейна рекурсивно, с заполнением кэша, а также вычисления расстояния Дамерау–Левенштейна рекурсивно и вычисления расстояния Дамерау–Левенштейна с заполнением кэша.

4 Экспериментальная часть

В данном разделе будет проведено функциональное тестирование разработанного программного обеспечения. Так же будет произведено измерение временных характеристик и характеристик по памяти каждого из реализованных алгоритмов.

Для проведения подобных экспериментов на языке программирования `Golang` [3], используется специальный пакет `testing` [4]. Данный пакет предоставляет инструменты для измерения процессорного времени и объема памяти, использованных конкретным алгоритмом в ходе проведения эксперимента.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- процессор: Intel Core™ i5-8250U [5] CPU @ 1.60GHz;
- память: 32 GiB;
- операционная система: Ubuntu [6] Linux [7] 20.04 64-bit;

Исследование 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.

4.2 Тестирование

В таблице 4.1 приведены функциональные тесты для алгоритмов сортировки.

Таблица 4.1 – Функциональные тесты

Тип случая	**Исходный массив**	**Пузырек**	**Вставками**	**Выбором**
пустой массив				
1 элемент в массиве	0	0	0	0
однотипные данные	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
худший	5 4 3 2 1	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5
лудший	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5
произвольный	7 3 -2 -8	-8 -2 3 7	-8 -2 3 7	-8 -2 3 7

При проведении функционального тестирования, полученные результаты работы программы совпали с ожидаемыми. Таким образом, функциональное тестирование пройдено успешно.

4.3 Временные характеристики

Для сравнения возьмем массивы размерностью [10, 100, 200, 300,…, 500, 1000]. Результаты замеров по результатам экспериментов приведены в Таблице 4.2.

Таблица 4.2 – Замер времени для массивов, размером от 10 до 1000 элементов

	Время, нс		
Длина (символ)	**Пузырьком**	**Вставками**	**Выбором**
10	44.14	13.04	66.98
100	3649	107.9	5942
200	13743	201.7	21728
300	28991	296.9	47210
400	50467	390.5	83345
500	77964	489.0	127831
1000	303805	969.4	507007

Отдельно сравним итеративные алгоритмы поиска расстояний Левенштейна и Дамерау–Левенштейна. Сравнение будет производится на основе данных, представленных в Таблице 4.2. Результат можно увидеть на Рисунке 4.1.

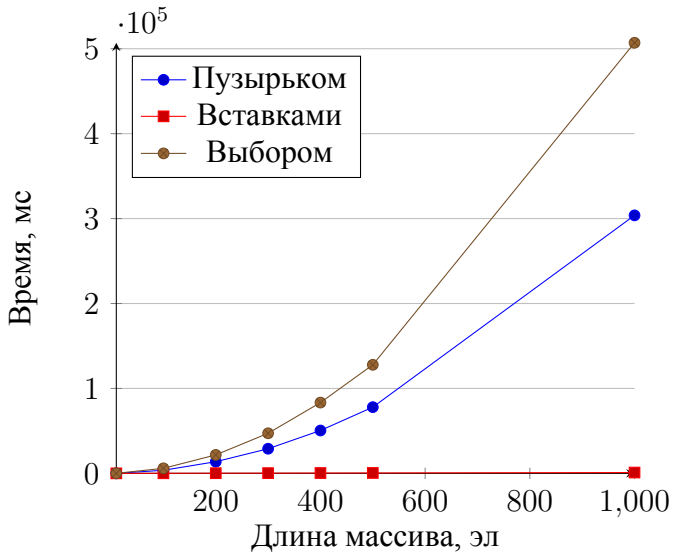

Рисунок 4.1 – Сравнение времени работы алгоритмов на отсортированных входных данных.

В лучшем случае, как и ожидалось, сортировка вставками является наиболее эффективной по времени. При этом, сортировка выбором является наименее эффективной, так так использует дополнительные операции, по сравнению с пузырьковой сортировкой.

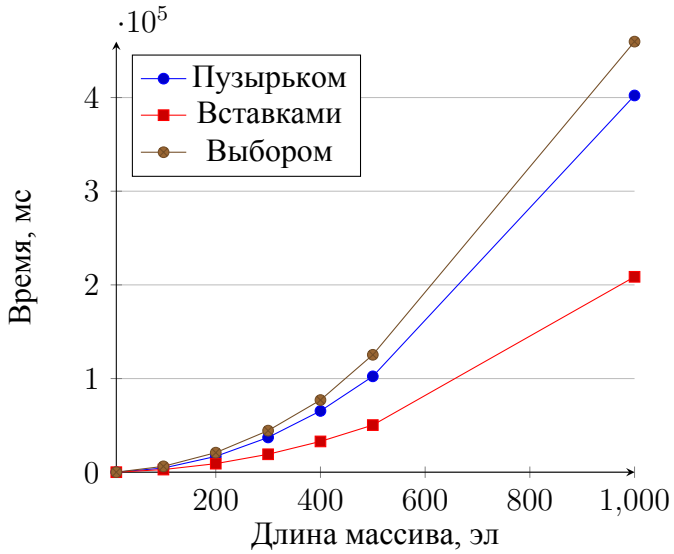

Рисунок 4.2 – Сравнение времени работы алгоритмов на отсортированных по убыванию входных данных.

В худшем случае сортировка вставками остается наиболее эффективной по времени из рассмотренных. При этом, сортировка пузырьком показывает результат близкий к сортировке выбором.

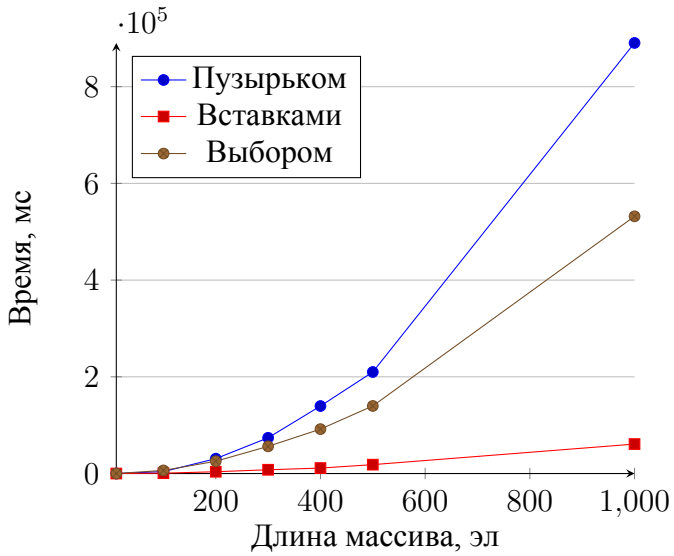

Рисунок 4.3 – Сравнение времени работы алгоритмов на случайно упорядоченных входных данных.

В общем случае сортировка вставками остается наиболее эффективной по времени из рассмотренных. При этом, сортировка пузырьком оказывается менее эффективной, так как в данном алгоритме элементы массива рассматриваются попарно и на каждой итерации происходит вплоть до N обменов элементов.

Из данных, приведенных в Таблице 4.2, видно, что сортировка вставками является наиболее эффективной по времени из рассмотренных в любом из трех случаев. При этом, сортировка пузырьком является практически эквивалентной сортировке выбором при входных

данных, отсортированных в обратном порядке, но оказывается наиболее затратной при случайном заполнении элементов в массиве.

4.4 Вывод

В данном разделе было произведено сравнение количества затраченного времени вышеизложенных алгоритмов.

Исходя из полученных результатов, можно сделать вывод, что сортировка пузырьком всегда оказывается самой времязатраной, сортировка вставками выигрывает по времени во всех трех случаях из трех представленных, а сортировка выбором оказывается эффективнее пузырьковой сортировки лишь на случайных данных.

Заключение

В данной работе было рассмотрено три алгоритма сортировки: вставками, пузырек и быстрая сортировка. Был описан и реализован каждый каждый алгоритм (листинги 3.3 – 3.5). Была произведена оценка трудоемкости реализаций алгоритмов сортировки. Также были показаны схемы работы алгоритмов (рисунки 2.1 – 2.3) Был выбраны и обоснованы средства реализации. А также приведены тесты (таблица 4.1).

В рамках выполнения работы решены следующие задачи.

- рассмотрены и изучены алгоритмы сортировки пузырьком, вставками и выбором;
- реализована каждая из этих сортировок;
- теоретически оценена их трудоемкость;
- проведено сравнение их временных характеристик экспериментально.

Литература

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

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

[3] The Go Programming Language [Электронный ресурс]. Режим доступа: https://golang.org/ (дата обращения: 14.09.2021).

[4] testing – The Go Programming Language [Электронный ресурс]. Режим доступа: https://golang.org/pkg/testing/ (дата обращения: 14.09.2021).

[5] Процессор Intel® Core™ i5-8250U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/124967/intel-core-i5-8250u-processor-6m-cache-up-to-3-40-ghz.html (дата обращения: 14.09.2021).

[6] Ubuntu: Enterprise Open Source and Linux [Электронный ресурс]. Режим доступа: https://ubuntu.com/ (дата обращения: 14.09.2021).

[7] LINUX.ORG.RU – Русская информация об ОС Linux [Электронный ресурс]. Режим доступа: https://www.linux.org.ru/ (дата обращения: 14.09.2021).