



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 Богаченко А.Е.

Группа ИУ7-56Б

Оценка (баллы) \_\_\_\_\_

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

Москва — 2020 г.

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Сортировка пузырьком . . . . .	4
1.2 Сортировка вставками . . . . .	4
1.3 Быстрая сортировка . . . . .	4
<b>2 Конструкторская часть</b>	<b>6</b>
2.1 Схемы алгоритмов . . . . .	6
2.2 Трудоемкость алгоритмов . . . . .	10
2.2.1 Сортировка вставками . . . . .	11
2.2.2 Сортировка пузырьком . . . . .	11
2.2.3 Быстрая сортировка . . . . .	12
<b>3 Технологическая часть</b>	<b>14</b>
3.1 Требования к ПО . . . . .	14
3.2 Средства реализации . . . . .	14
3.3 Листинг кода . . . . .	15
<b>4 Исследовательская часть</b>	<b>17</b>
4.1 Пример работы . . . . .	17
4.2 Технические характеристики . . . . .	17
4.3 Время выполнения алгоритмов . . . . .	17
4.4 Производительность алгоритмов . . . . .	19
<b>Заключение</b>	<b>21</b>
<b>Литература</b>	<b>22</b>

# Введение

На сегодняшний день существует не одна вариация алгоритмов сортировки. Все они различаются по скорости сортировки и по объему необходимой памяти.

Цель данной лабораторной работы - обучение расчету трудоемкости алгоритмов

Алгоритмы сортировки часто применяются в практике программирования. В том числе в областях связанных с математикой, физикой, компьютерной графикой и т.д.

В ходе лабораторной работы предстоит:

- изучить алгоритмы сортировок;
- дать теоретическую оценку алгоритмам сортировки;
- реализовать три алгоритма сортировки на одном из языков программирования;
- сравнить алгоритмы сортировки.

# 1 Аналитическая часть

## 1.1 Сортировка пузырьком

Алгоритм [1] состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов.

## 1.2 Сортировка вставками

На каждом шаге выбирается один из элементов неотсортированной части массива (максимальный/минимальный) [1] и помещается на нужную позицию в отсортированную часть массива.

## 1.3 Быстрая сортировка

Общая идея [2] алгоритма состоит в следующем:

1. выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность;
2. сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньше опорного», «равные» и «большие»;
3. для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм деления [3].

## Вывод

Были рассмотрены три алгоритма сортировки.

## 2 Конструкторская часть

### 2.1 Схемы алгоритмов

На рисунке 2.1 приведена схема классического алгоритма сортировки пузырьком.

На рисунке 2.2 приведена схема алгоритма сортировки вставками.

На рисунке 2.3 приведена схема быстрой сортировки вместе с алгоритмом разделения 2.4.

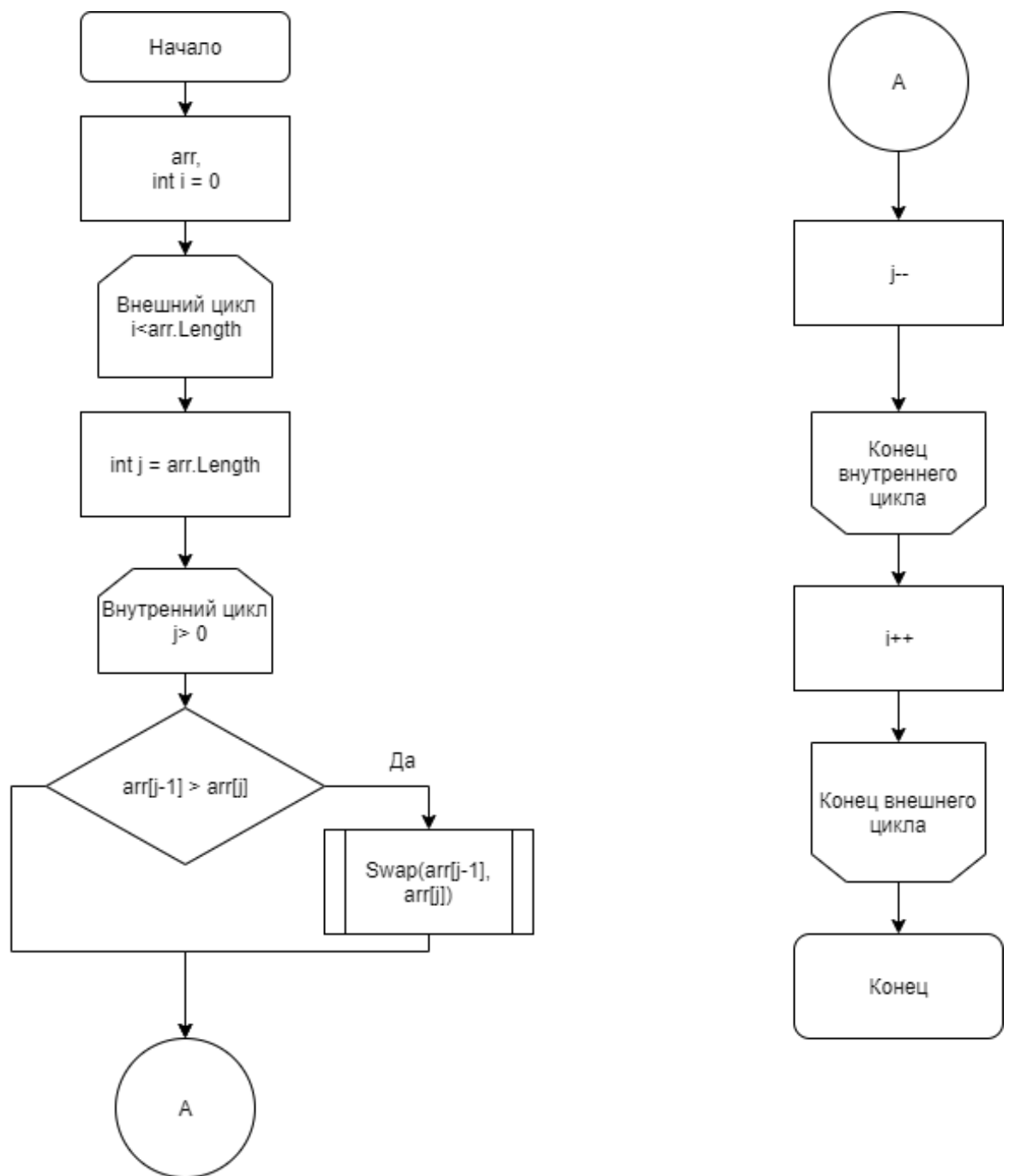


Рисунок 2.1 – Схема алгоритма сортировки пузырьком

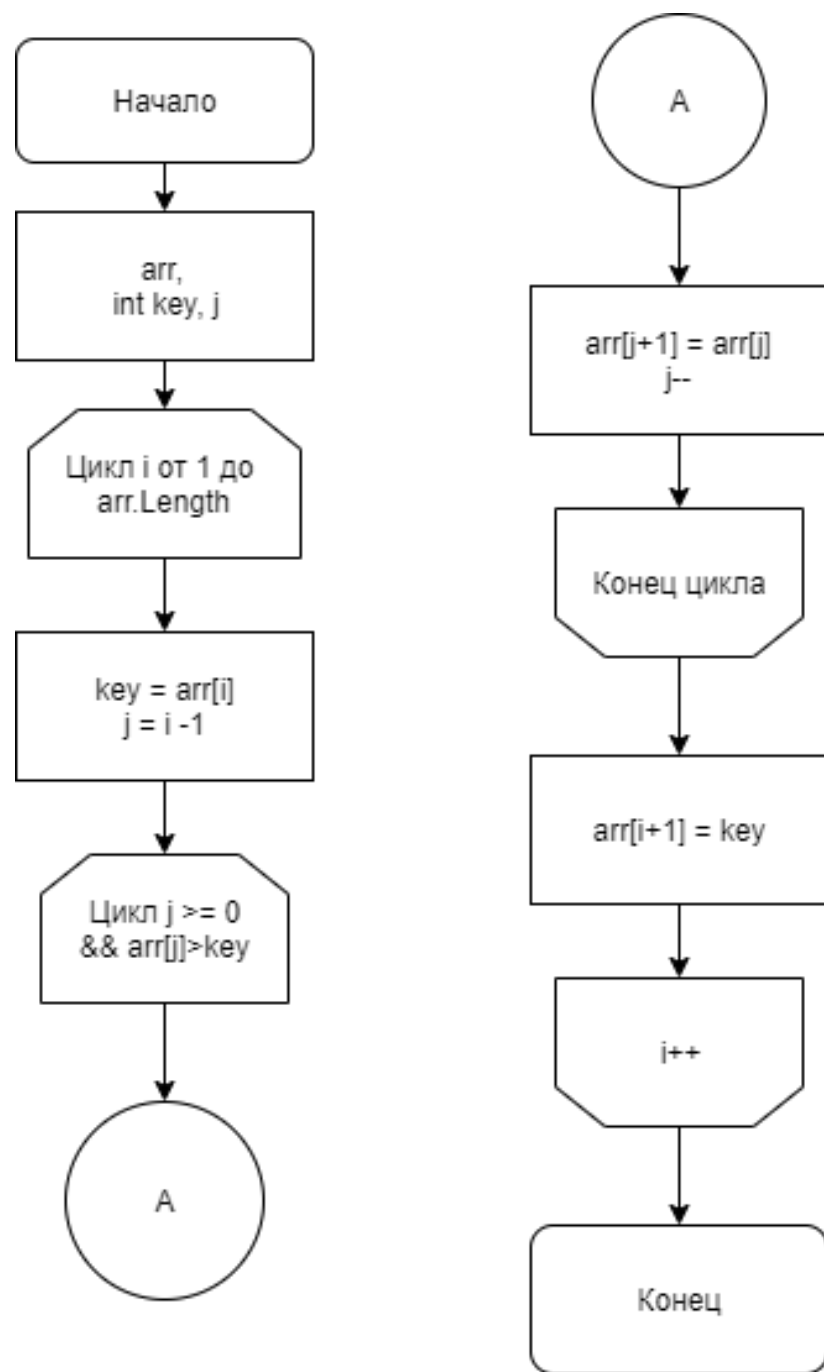


Рисунок 2.2 – Схема алгоритма сортировки вставками



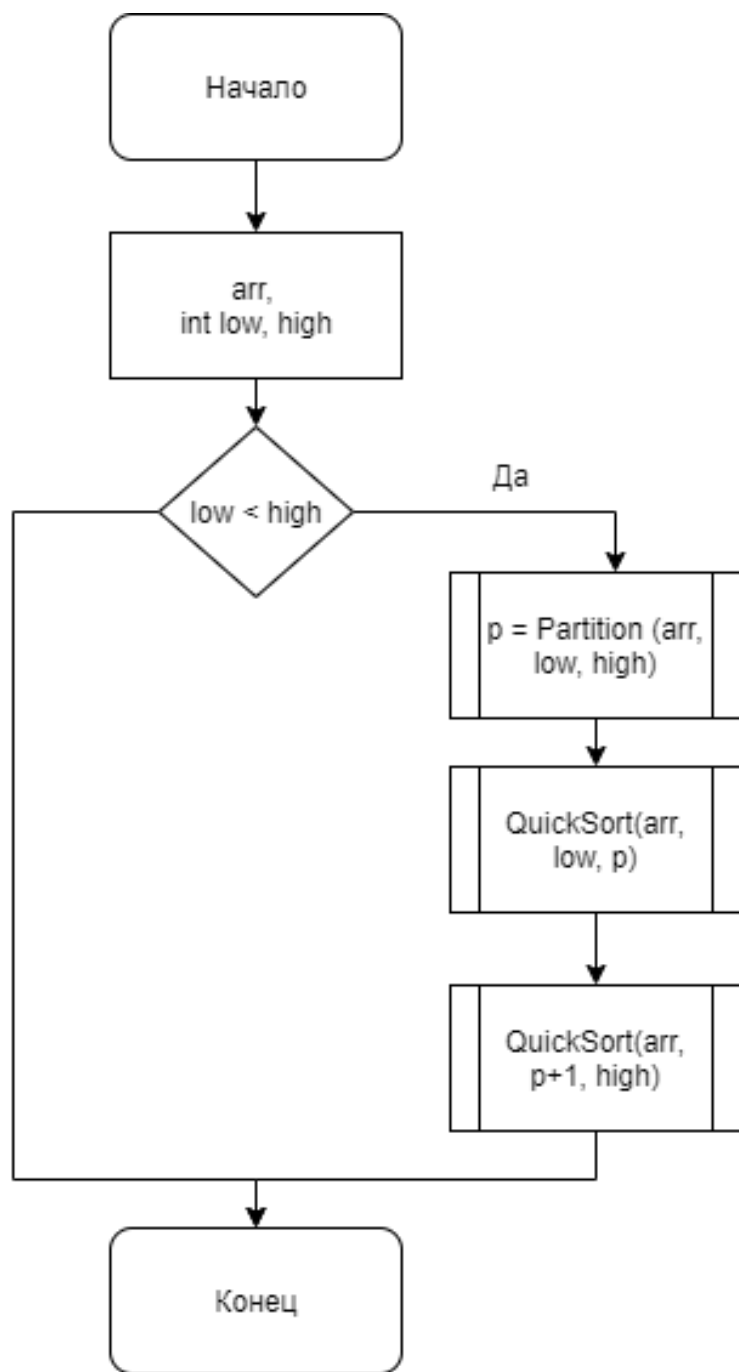


Рисунок 2.3 – Схема алгоритма быстрой сортировки

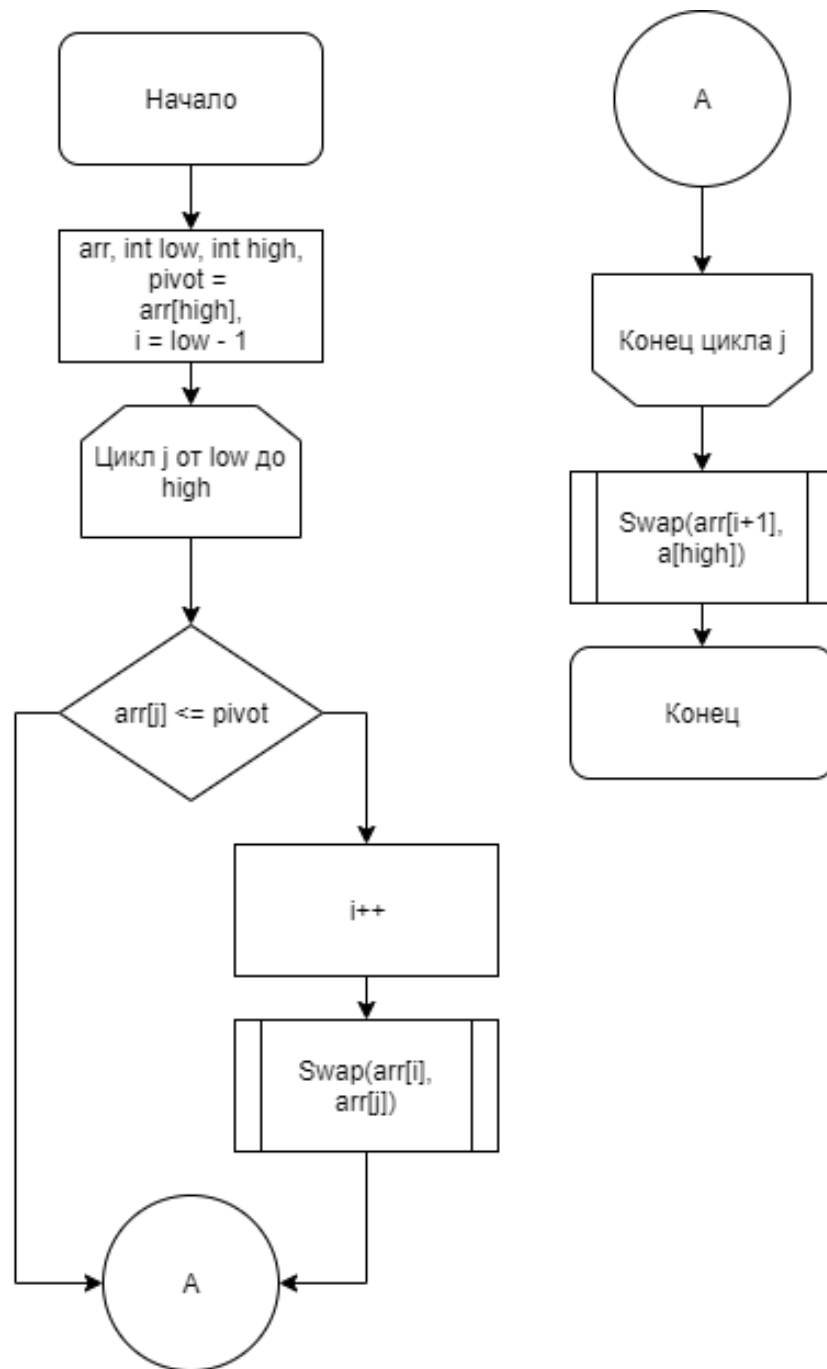


Рисунок 2.4 – Схема алгоритма разделения

## 2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1 —  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $=$ ,  $==$ ,  $<=$ ,  $>=$ ,  $!=$ ,  $+=$ ,  $-=$ ,  $++$ ,  $--$  — получение полей класса;

- оценка трудоемкости цикла:  $F_{\text{ц}} = a + N^*(a + F_{\text{тела}})$ , где  $a$  - условие цикла;
- стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

### 2.2.1 Сортировка вставками

В таблице 2.1 приведена построчная оценка трудоемкости алгоритма сортировки вставками.

Таблица 2.1 – Построчная оценка веса

Код	Вес
for i in range(1, len(a)):	$1+n*2$
key = a[i]	2
j = i-1	2
while (j >= 0 and a[j] > key):	$n*4$
a[j+1] = a[j]	4
j-= 1	1
a[j+1] = key	3

**Лучший случай:** отсортированный массив. При этом все внутренние циклы состоят всего из одной итерации.

Трудоемкость:  $T(n) = 1 + 2n * (2 + 2 + 3) = 2n * 7 = 14n + 1 = O(n)$

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

Трудоемкость:  $T(n) = 1 + n * (2 + 2 + 4n * (4 + 1) + 3) = 2n * n + 7n + 1 = O(n^2)$

### 2.2.2 Сортировка пузырьком

**Лучший случай:** Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла

Трудоемкость:  $1 + 2n * (1 + 2n * 4) = 1 + 2n + 16n * n = O(n^2)$

**Худший случай:** Массив отсортирован в обратном порядке; в каждом случае происходил обмен

Трудоемкость:  $1 + 2n * (1 + 2n * (4 + 5)) = O(n^2)$

### 2.2.3 Быстрая сортировка

**Лучший случай:** сбалансированное дерево вызовов<sup>[2]</sup>  $O(n * \log(n))$

В наиболее благоприятном случае процедура PARTITION приводит к двум подзадачам, размер каждой из которых не превышает  $\frac{n}{2}$ , поскольку размер одной из них равен  $\frac{n}{2}$ , а второй  $\frac{n}{2} - 1$ . В такой ситуации быстрая сортировка работает намного производительнее, и время ее работы описывается следующим рекуррентным соотношением:  $T(n) = 2T(\frac{n}{2}) + O(n)$ , где мы не обращаем внимания на неточность, связанную с игнорированием функций “пол” и “потолок”, и вычитанием 1. Это рекуррентное соотношение имеет решение ;  $T(n) = O(n \lg n)$ . При сбалансированности двух частей разбиения на каждом уровне рекурсии мы получаем асимптотически более быстрый алгоритм.

Фактически любое разбиение, характеризующееся конечной константой пропорциональности, приводит к образованию дерева рекурсии высотой  $O(\lg n)$  со стоимостью каждого уровня, равной  $O(n)$ . Следовательно, при любой постоянной пропорции разбиения полное время работы быстрой сортировки составляет  $O(n \lg n)$ .

**Худший случай:** несбалансированное дерево<sup>[2]</sup>  $O(n^2)$  Поскольку рекурсивный вызов процедуры разбиения, на вход которой подается массив размером 0, приводит к немедленному возврату из этой процедуры без выполнения каких-либо операций,  $T(0) = O(1)$ . Таким образом, рекуррентное соотношение, описывающее время работы процедуры в указанном случае, записывается следующим образом:  $T(n) = T(n-1) + T(0) + O(n) = T(n-1) + O(n)$ . Интуитивно понятно, что при суммировании промежутков времени, затрачиваемых на каждый уровень рекурсии, получается арифметическая прогрессия, что приводит к результату  $O(n^2)$ .

## Вывод

На основе теоретических данных, полученных из аналитического раздела были построены схемы требуемых алгоритмов, были установлены лучшие и худшие случаи для алгоритмов сортировок.

Сортировка пузырьком: лучший -  $O(n)$ , худший -  $O(n^2)$

Сортировка вставками: лучший -  $O(n)$ , худший -  $O(n^2)$

Быстрая сортировка: лучший -  $O(n \lg n)$ , худший -  $O(n^2)$

## 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:

- корректная сортировка.

### 3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран многопоточный язык GO [4]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка. Так же данный язык предоставляет средства тестирования разработанного ПО.

## 3.3 Листинг кода

В листингах 3.1–3.3 приведены реализации алгоритмов умножения матриц.

Листинг 3.1 – Сортировка пузырьком

```
1  for range a {
2      for i := len(a) - 1; i > 0; i-- {
3          if a[i-1] > a[i] {
4              a[i-1], a[i] = a[i], a[i-1]
5          }
6      }
7  }
8  }
```

Листинг 3.2 – Сортировка вставками

```
1  for i := 1; i < len(a); i++ {
2      key := a[i]
3      j := i - 1
4      for j >= 0 && a[j] > key {
5          a[j+1] = a[j]
6          j--
7      }
8      a[j+1] = key
9  }
10 }
```

Листинг 3.3 – Быстрая сортировка

```
1  func QuickSort(a []int) []int {
2      if len(a) < 2 {
3          return a
4      }
5
6      left, right := 0, len(a)-1
7
8      pivotIndex := rand.Int() % len(a)
9
10     a[pivotIndex], a[right] = a[right], a[pivotIndex]
11
12     for i := range a {
13         if a[i] < a[right] {
14             a[i], a[left] = a[left], a[i]
15             left++
16         }
17     }
```

```

18
19     a[left], a[right] = a[right], a[left]
20
21     QuickSort(a[:left])
22     QuickSort(a[left+1:])
23
24     return a
25 }

```

В таблице 3.1 приведены функциональные тесты для алгоритмов сортировок.

Таблица 3.1 – Функциональные тесты

Исходные данные	Ожидаемый результат
1 2 3 4 5	1 2 3 4 5
5 4 3 2 1	1 2 3 4 5
2 4 5 3 1	1 2 3 4 5

## Вывод

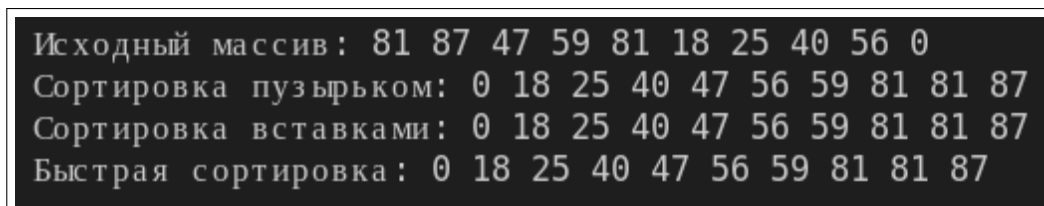
Были разработаны и протестированы спроектированные алгоритмы: сортировки пузырьком, сортировки вставками и быстрой сортировки.



## 4 Исследовательская часть

### 4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.



```
Исходный массив: 81 87 47 59 81 18 25 40 56 0
Сортировка пузырьком: 0 18 25 40 47 56 59 81 81 87
Сортировка вставками: 0 18 25 40 47 56 59 81 81 87
Быстрая сортировка: 0 18 25 40 47 56 59 81 81 87
```

Рисунок 4.1 – Демонстрация работы алгоритмов сортировки

### 4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Kali [5] Linux [6] 5.8.10-1kali1 64-bit.
- Память: 8 GB.
- Процессор: Intel® Core™ i5-8250U [7] CPU @ 1.60GHz

Тестирование проводилось на ноутбуке при включённом режиме производительности. Во время тестирования ноутбук был нагружен только системными процессами.

### 4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи написания «бенчмарков» [8], предоставляемых 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

```
1 // Special case benchs
2
3 func BenchmarkBubble1(b *testing.B) {
4     a := Generate(1);
5
6     for i := 0; i < b.N; i++ {
7         BubbleSort(a)
```

На рисунках 4.2 — 4.4 приведены графики зависимостей времени работы алгоритмов от различных наборов данных.

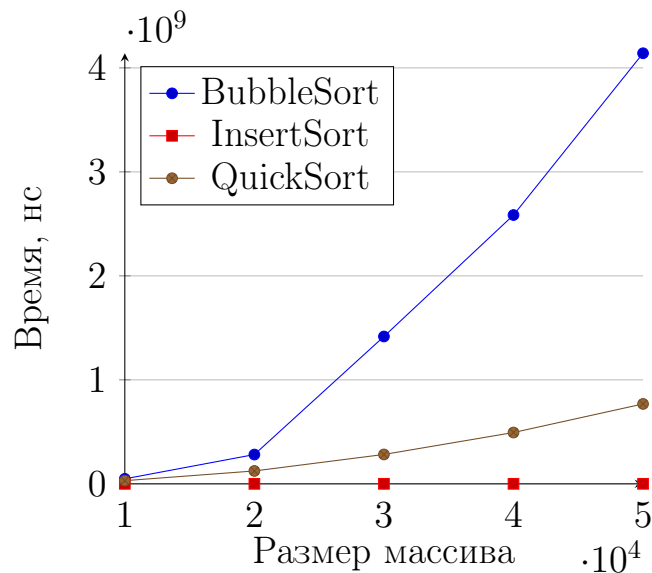


Рисунок 4.2 – Сравнение алгоритмов на массивах неупорядоченных данных.

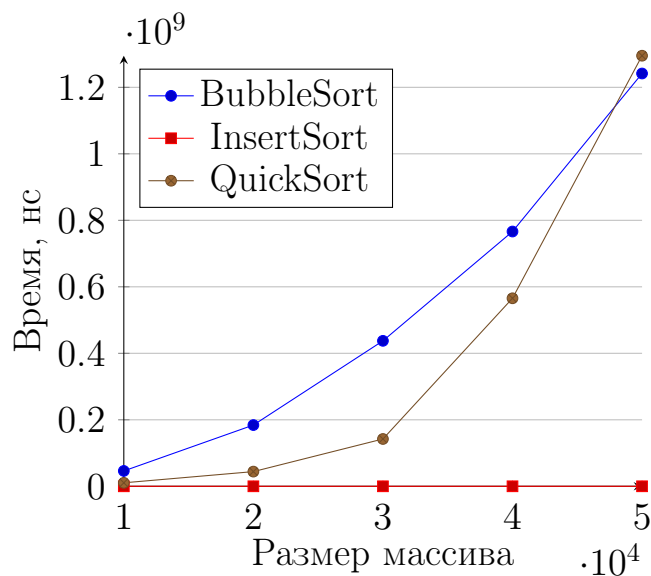


Рисунок 4.3 – Сравнение алгоритмов на массивах упорядоченных по возрастанию данных.

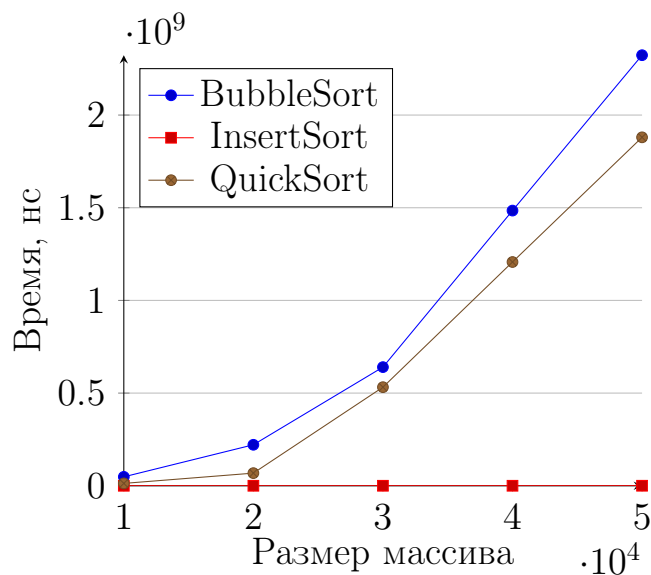


Рисунок 4.4 – Сравнение алгоритмов на массивах упорядоченных по убыванию данных.

## 4.4 Производительность алгоритмов

Производительность и объем выделенной памяти при работе алгоритмов указаны на рисунке 4.5.

```

goos: linux
goarch: amd64
BenchmarkBubble10k-8      24      48976892 ns/op      3413 B/op      0 allocs/op
BenchmarkBubble50k-8       1     4131619453 ns/op     401408 B/op     1 allocs/op
BenchmarkBubble100k-8      1     17240296778 ns/op     802816 B/op     1 allocs/op
BenchmarkInsert10k-8     131580      9020 ns/op         0 B/op      0 allocs/op
BenchmarkInsert50k-8     16627      68098 ns/op        24 B/op      0 allocs/op
BenchmarkInsert100k-8      1    1547841966 ns/op     802816 B/op     1 allocs/op
BenchmarkQuick10k-8       100     31209606 ns/op       819 B/op      0 allocs/op
BenchmarkQuick50k-8       32     744918100 ns/op     12544 B/op     0 allocs/op
BenchmarkQuick100k-8      12    2657977577 ns/op     66901 B/op     0 allocs/op
PASS
ok      _/root/bmstu-aa/lab3/src/sort 93.231s

```

Рисунок 4.5 – Замеры производительности алгоритмов, выполненные при помощи команды `go test -bench . -benchmem`

## Вывод

Были протестированы различные алгоритмы сортировок. По результатам эксперимента алгоритм сортировки пузырьком показывает худшие временные показатели. Алгоритм быстрой сортировки показал худший результат на упорядоченных наборах данных. Самые лучшие временные показатели во всех категориях показал алгоритм сортировки вставками.

# Заключение

В ходе выполнения данной лабораторной работы были реализованы три алгоритма сортировки: сортировка пузырьком, сортировка вставками и быстрая сортировка. Был проведён анализ каждого алгоритма и измерено время работы алгоритмов для массивов разных размеров. Была оценена трудоёмкость алгоритмов.

Лучшее время работы показала сортировка вставками, худшее – сортировка пузырьком.

# Литература

- [1] Кнут Дональд. Искусство программирования. Том 3. Сортировка и поиск. – Диалектика-Вильямс, 1998.
- [2] А.В.Левитин. Алгоритмы: построение и анализ. – М. Вильямс. Глава 4. Метод декомпозиции: Быстрая сортировка, 2006.
- [3] Т. Кормен Ч. Лейзерсон Р. Ривест К. Штайн. Алгоритмы: построение и анализ. – Под. ред. И. В. Красикова – 2-е издание. Глава 7. Быстрая сортировка, 2005.
- [4] The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/> (дата обращения: 09.09.2020).
- [5] Our Most Advanced Penetration Testing Distribution, Ever. [Электронный ресурс]. Режим доступа: <https://kali.org/> (дата обращения: 12.09.2020).
- [6] Linux – Википедия [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Linux> (дата обращения: 12.09.2020).
- [7] Intel Processors [Электронный ресурс]. Режим доступа: <https://www.intel.com/content/www/us/en/products/processors/core/i5-processors.html> (дата обращения: 12.09.2020).
- [8] testing – The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/pkg/testing/> (дата обращения: 12.09.2020).