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 Лемешев А. П.

Группа ИУ7-52Б

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

Москва — 2022 г.

Оглавление

Введение

Сортировка – это процесс разделения объектов по виду или сорту, программисты традиционно используют это слово в гораздо более узком смысле, обозначая им такую перестановку предметов, при которой они располагаются в порядке возрастания или убывания[1]. Такой процесс следовало бы называть не сортировкой, а *упорядочением*, но использование этого слова привело бы к путанице из-за перегруженности значениями слова *порядок*.

Алгоритмы сортировки используются практически в любой программной системе. Целью алгоритмов сортировки является упорядочение последовательности элементов данных. Поиск элемента в последовательности отсортированных данных занимает время, пропорциональное логарифму количеству элементов в последовательности, а поиск элемента в последовательности не отсортированных данных занимает время, пропорциональное количеству элементов в последовательности, то есть намного больше. Существует множество различных методов сортировки данных. Однако любой алгоритм сортировки можно разбить на три основные части:

- сравнение, определяющее упорядоченность пары элементов;
- перестановка, меняющая местами пару элементов;
- собственно сортирующий алгоритм, который осуществляет сравнение и перестановку элементов данных до тех пор, пока все эти элементы не будут упорядочены.

Важнейшей характеристикой любого алгоритма сортировки является скорость его работы, которая определяется функциональной зависимостью среднего времени сортировки последовательностей элементов данных, заданной длины, от этой длины. Время сортировки будет пропорционально количеству сравнений и перестановки элементов данных в процессе их сортировки.

Задачи лабораторной работы:

- изучить и реализовать 3 алгоритма сортировки: расчёской, вставками, блинная;
- провести сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- провести сравнительный анализ алгоритмов на основе экспериментальных данных;
- подготовить отчета по лабораторной работе.

1 Аналитическая часть

1.1 Сортировка расчёской

Сортировка расчёской улучшает сортировку пузырьком, и конкурирует с алгоритмами, подобными быстрой сортировке. Основная идея — устранить черепах, или маленькие значения в конце списка, которые крайне замедляют сортировку пузырьком (кролики, большие значения в начале списка, не представляют проблемы для сортировки пузырьком). В сортировке пузырьком, когда сравниваются два элемента, промежуток (расстояние друг от друга) равен 1. Основная идея сортировки расчёской в том, что этот промежуток может быть гораздо больше, чем единица.

1.2 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.

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 блинная

Единственная операция, допустимая в алгоритме сортировки — переворот элементов последовательности до какого-либо индекса. В отличие от традиционных алгоритмов, в которых минимизируют количество сравнений, в блинной сортировке требуется сделать как можно меньше перево-

ротов. Процесс можно визуально представить как стопку блинов, которую тасуют путём взятия нескольких блинов сверху и их переворачивания.

Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: расчёской, вставками и блинная. Необходимо оценить теоретическую оценку алгоритмов и проверить ее экспериментально.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены схемы алгоритмов сортировки расчёской, вставками и блинной соответственно. На рисунке 2.4 представлены схемы дополнительных алгоритмов, использующихся в блинной сортировке.

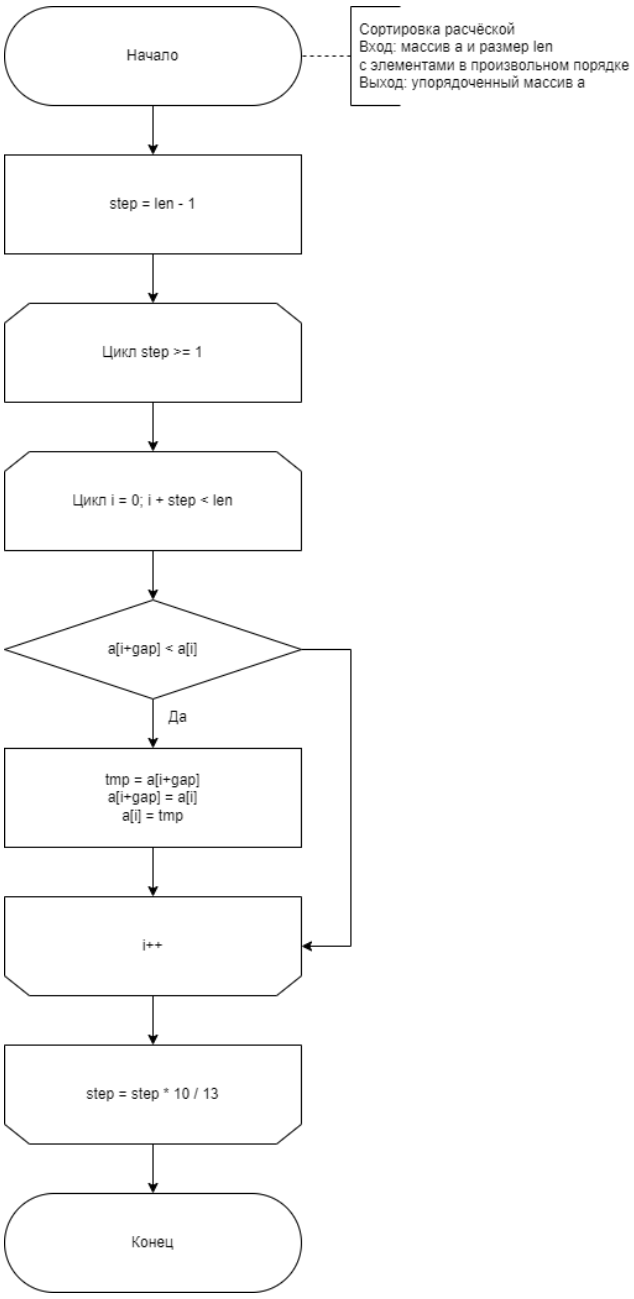

Рис. 2.1: Схема алгоритма сортировки расчёской

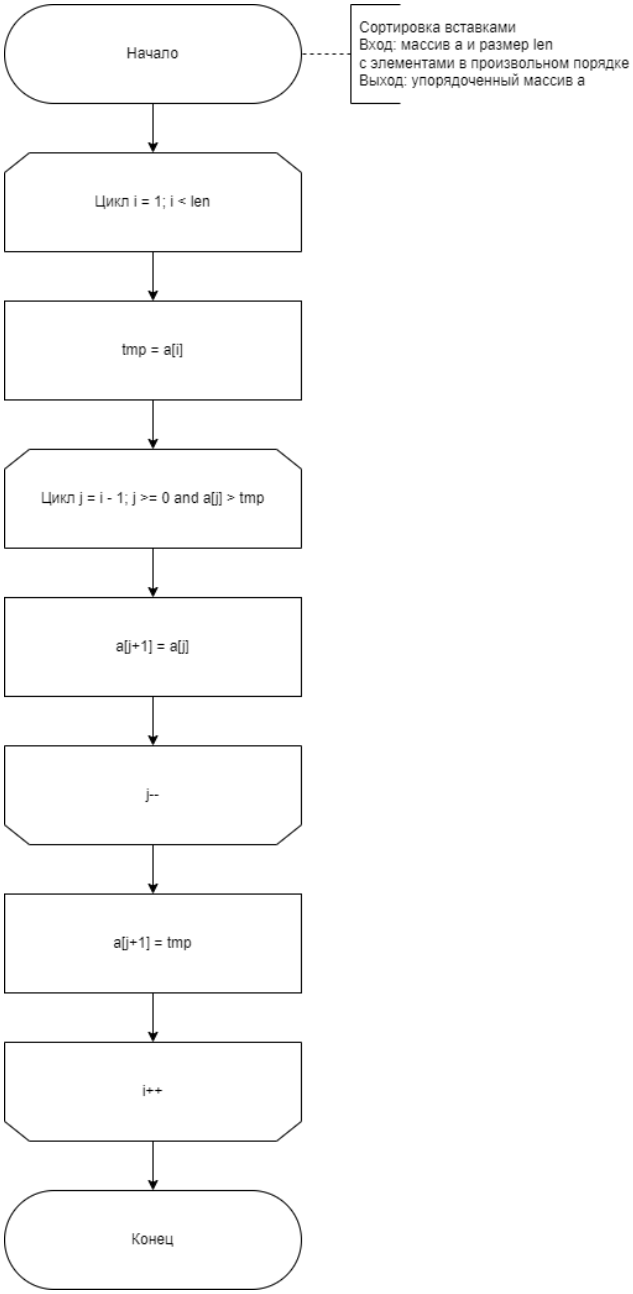

Рис. 2.2: Схема алгоритма сортировки вставками

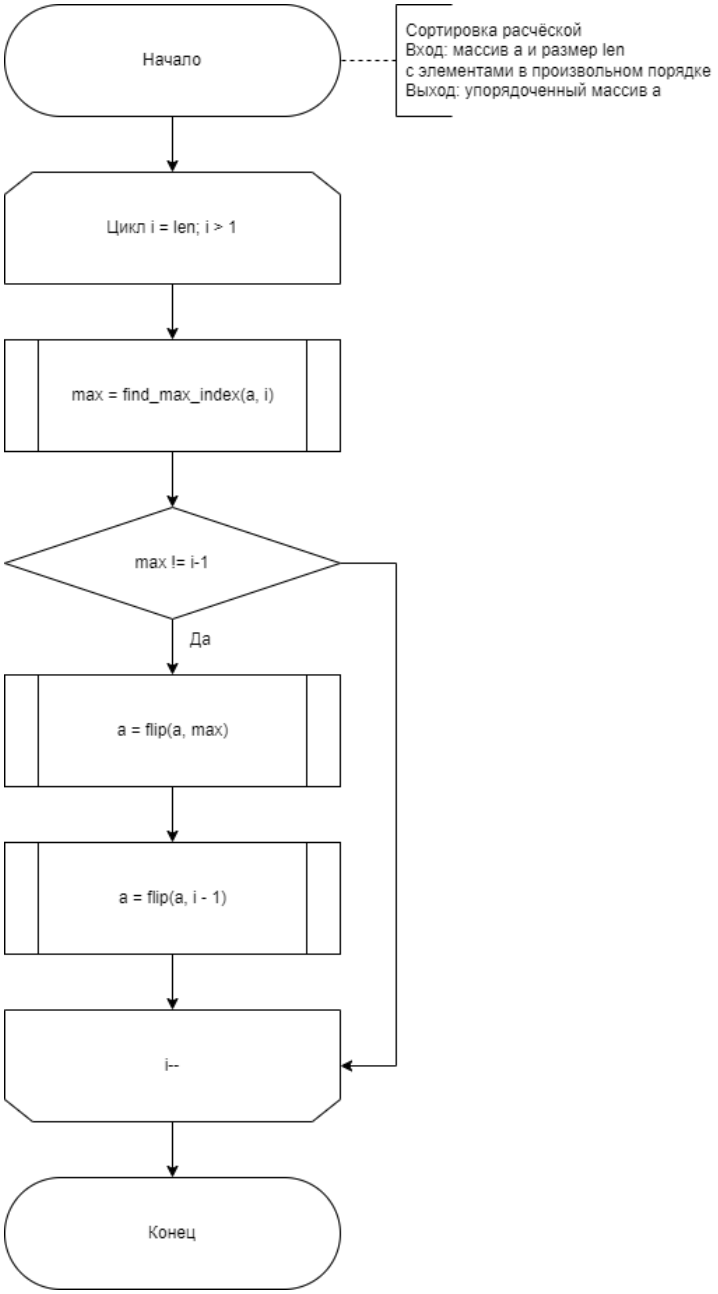

Рис. 2.3: Схема алгоритма сортировки блинной

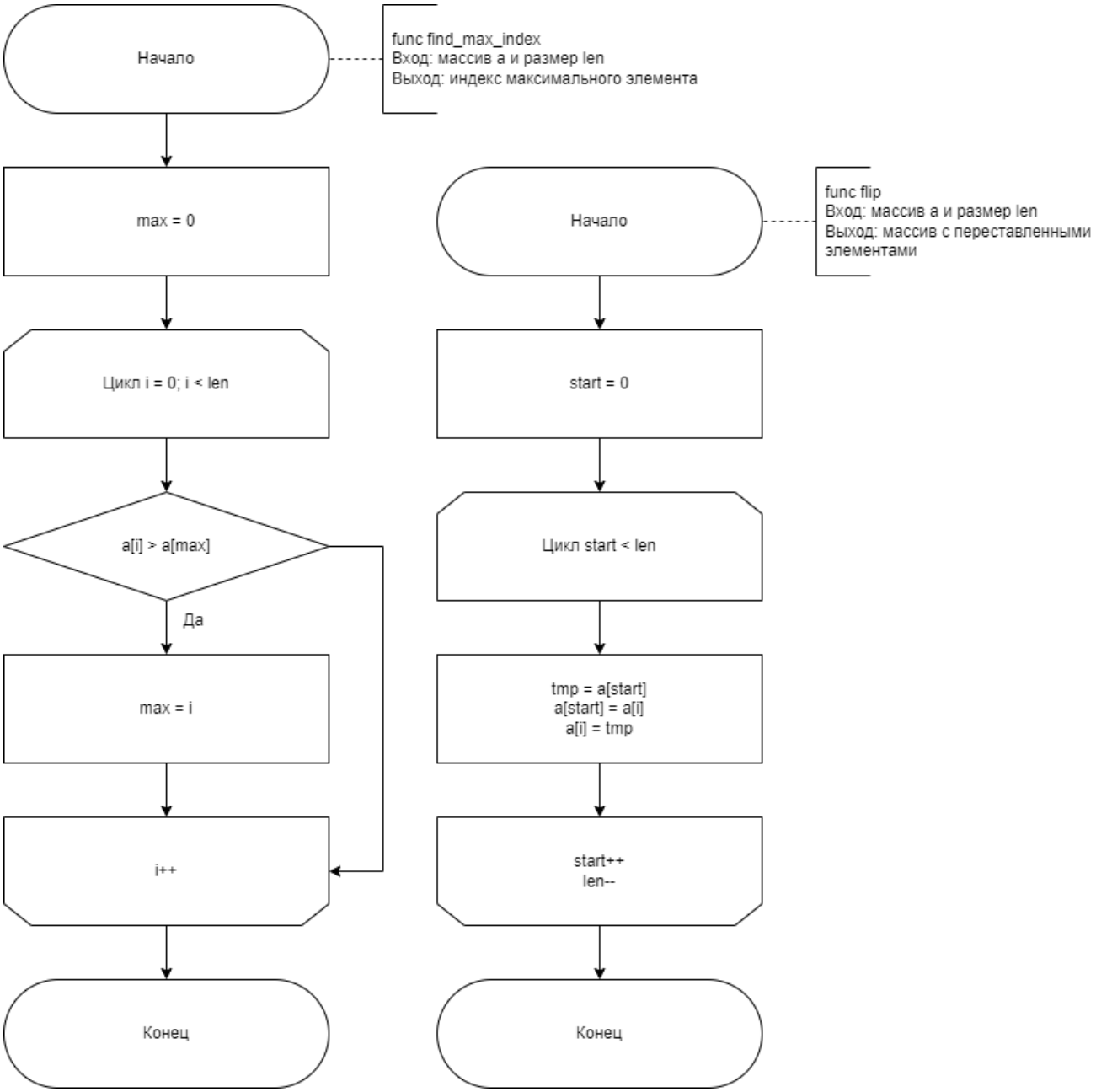

Рис. 2.4: Схемы алгоритмов поиска максимального индекса и перестановки элементов в массиве

2.2 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. операции из списка (2.1) имеют трудоемкость 1;

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

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

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

3. трудоемкость цикла рассчитывается, как (2.3);

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

4. трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

Обозначим во всех последующих вычислениях размер массивов как N.

2.3.1 Алгоритм сортировки расчёской

Трудоёмкость алгоритма сортировки расчёской состоит из:

- Трудоёмкость сравнения и инкремента внешнего цикла, которая равна (2.4):

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

- Трудоёмкость внутреннего цикла, количество итераций которых меняется в промежутке $[0..N - step]$, которая равна (2.5):

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

- Трудоёмкость условия во внутреннем цикле, которая равна (2.6):

$$f_{if} = 4 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases} \tag{2.6}$$

Трудоёмкость в лучшем случае (2.7):

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

Трудоёмкость в худшем случае (2.8):

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

2.3.2 Алгоритм сортировки вставками

Трудоёмкость алгоритма сортировки вставками состоит из:

- Трудоёмкость сравнения, инкремента внешнего цикла, а также зависимых только от него операций, по $i \in [1..N)$, которая равна (2.9):

$$f_{outer} = 1 + 3(N-1) \tag{2.9}$$

- Трудоёмкость внутреннего цикла, количество итераций которого меняется в промежутке $[1..N-1]$, которая равна (2.10):

$$f_{inner} = 7(N-1) + N \cdot (N-1) \cdot f_{if} \tag{2.10}$$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.11):

$$f_{if} = 3 + \begin{cases} 0, & \text{л.с.} \\ 5, & \text{х.с.} \end{cases} \tag{2.11}$$

Трудоёмкость в лучшем случае (2.12):

$$f_{best} = 1 + 3N - 3 + 7N - 7 \approx 10N = O(N) \tag{2.12}$$

Трудоёмкость в худшем случае (2.13):

$$f_{worst} = 1 + 3N - 3 + 7N - 7 + 5N^2 - 5N \approx 5N^2 = O(N^2) \tag{2.13}$$

2.3.3 Алгоритм блинной сортировки

Трудоёмкость алгоритма блинной сортировки состоит из:

- Трудоёмкость сравнения, инкремента внешнего цикла, а также зависимых только от него операций, по $i \in [0..N)$, которая равна (2.14):

$$f_{outer} = 1 + 2(N - 1) + 3(N - 1) \tag{2.14}$$

- Трудоёмкость условия во внешнем цикле, которая равна (2.15):

$$f_{if} = 3 + \begin{cases} 0, & \text{л.с.} \\ f_{inner}, & \text{х.с.} \end{cases} \tag{2.15}$$

- Трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N - k]$ и $[k..N]$, которая равна (2.16):

$$f_{inner} = 7(N - 1) + 2(N) \tag{2.16}$$

Трудоёмкость в лучшем случае (2.17):

$$f_{best} = 1 + 3N - 3 + 7N - 7 \approx 10N = O(N) \tag{2.17}$$

Трудоёмкость в худшем случае (2.18):

$$f_{worst} = 1 + 3N - 3 + 7N - 7 + 5N^2 - 5N \approx 5N^2 = O(N^2) \qquad (2.18)$$

Вывод

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

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся массив сравнимых элементов;
- на выходе — тот же массив, но в отсортированном порядке.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран современный компилируемый ЯП Golang [2]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка, а также тем, что данный язык предоставляет широкие возможности для написания тестов [3].

3.3 Листинг кода

В листингах 3.1 – 3.3 приведены листинги алгоритма сортировки расчёской, вставками и блинной соответственно. В листингах 3.4 – 3.5 приведены примеры реализации тестов и бенчмарков.

Листинг 3.1: Алгоритм сортировки расчёской

```
func (a Array) SortComb() {
	step := a.Len() - 1;

	for step >= 1 {
		for i := 0; i + step < a.Len(); i++ {
			if (a.Less(i + step, i)) { a.Swap(i, i + step) }
		}

		step = step * 10 / 13;
	}
}
```

Листинг 3.2: Алгоритм сортировки вставками

```

func (a Array) SortInsert() {
	for i := 1; i < a.Len(); i++ {
		key := a[i]
		j := i - 1

		for j >= 0 && a[j] > key {
			a[j + 1] = a[j]
			j--
		}

		a[j + 1] = key
	}
}
```

Листинг 3.3: Алгоритм блинной сортировки

```
func (a Array) SortPancake() {
	for cur_n := a.Len(); cur_n > 1; cur_n-- {
		i := a.findMaxIndex(cur_n)

		if i != cur_n-1 {
			a.flip(i)
			a.flip(cur_n - 1)
		}
	}
}

func (a Array) findMaxIndex(len int) int {
	max_i := 0

	for i := 0; i < len; i++ {
		if a.Less(max_i, i) { max_i = i }
	}

	return max_i
}

func (a Array) flip(i int) {
	start := 0

	for start < i {
		a.Swap(start, i)
		start++
		i--
	}
}
```

Листинг 3.4: Пример реализации теста

```
var testSortTable = []struct {
        title string
        in Array
        out Array
} {
        {
                title: "default",
                in: Array{1, 3, 5, 2, 4},
                out: Array{1, 2, 3, 4, 5},
        },
        {
                title: "asc",
                in: Array{1, 2, 3, 4, 5},
                out: Array{1, 2, 3, 4, 5},
        },
}

func TestSortComb(t *testing.T) {
        for _, test := range testSortTable {
                // Arrange
                arr := test.in.Copy()

                // Act
                t.Logf("starting␣test␣'%v'\n", test.title)
                arr.SortComb()

                // Assert
                if !arr.Compare(test.out) {
                        t.Errorf("Incorrect␣result.\ntitle:␣%v\nin:␣%v\nout:␣%v\nres:␣
                            %v\n",
                                test.title, test.in, test.out, arr)
                }
        }
}
```

Листинг 3.5: Пример реализации бенчмарка

```
func benchmark(arr array.Array) error {
	var (
		durations [4]time.Duration
		arrs [4]array.Array
		repeats = *repeatsFlag
	)

	for i := 0; i < repeats; i++ {
		arrs[0] = arr.Copy()
		arrs[1] = arr.Copy()
		arrs[2] = arr.Copy()
		arrs[3] = arr.Copy()

		start := time.Now()
		sort.Sort(arrs[0])
		durations[0] += time.Since(start)

		start = time.Now()
		arrs[1].SortComb()
		durations[1] += time.Since(start)

		start = time.Now()
		arrs[2].SortInsert()
		durations[2] += time.Since(start)

		start = time.Now()
		arrs[3].SortPancake()
		durations[3] += time.Since(start)
	}

	if !array.Compare(arrs[0], arrs[1], arrs[2], arrs[3]) {
		return fmt.Errorf("arrays not equal after sort")
	}

	return nil
}
```

3.4 Тестирование функций

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

Таблица 3.1: Тестовые данные

Входной массив	Ожидаемый результат	Результат
[1,3,5,2,4]	[1,2,3,4,5]	[1,2,3,4,5]
[1,2,3,4,5]	[1,2,3,4,5]	[1,2,3,4,5]
[5,4,3,2,1]	[1,2,3,4,5]	[1,2,3,4,5]
[5, − 1,4,3,2]	[−1,2,3,4,5]	[−1,2,3,4,5]
[777]	[777]	[777]
[]	[]	[]

Вывод

Правильный выбор инструментов разработки позволил эффективно реализовать алгоритмы, настроить модульное тестирование и выполнить исследовательский раздел лабораторной работы.

4 Исследовательская часть

4.1 Технические характеристики

- Операционная система: Windows 11 x64[4].
- Память: 8 GiB.
- Процессор: AMD Ryzen 5 3550H[5].

Замеры проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

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

Размер	Время сортировки, нс		
	Расчёской	**Вставками**	**Блинная**
100	1065	167	5570
200	3412	456	26224
300	5377	692	52727
400	8469	779	96144
500	9234	865	139676
1000	22264	1607	591937
1500	33775	2727	1286341
2000	55982	3742	2585800
2500	70644	4910	4074924
5000	137710	8256	13772451
10000	311604	16428	54550642

Таблица 4.1: Время работы алгоритмов сортировки на отсортированных данных

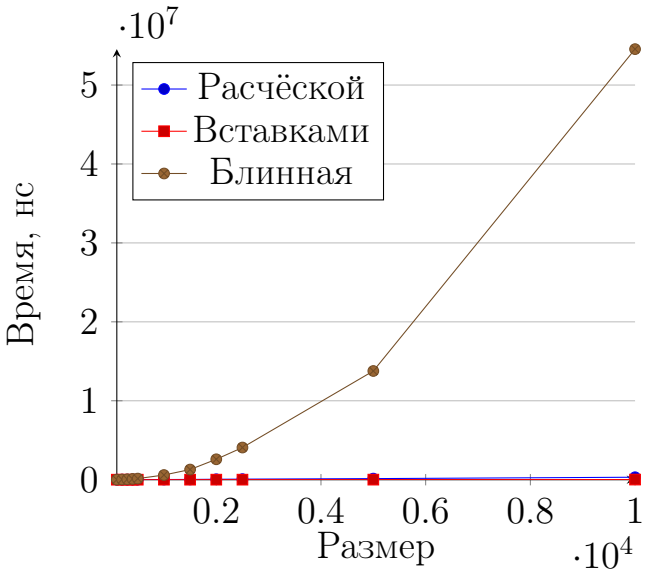

Рис. 4.1: Зависимость времени работы алгоритма сортировки от размера отсортированного массива

Размер	Время сортировки, нс		
	Расчёской	**Вставками**	**Блинная**
100	1883	6267	7727
200	4263	22956	29378
300	6990	51433	67066
400	10120	92757	118769
500	12771	137380	177794
1000	23620	509311	673713
1500	46180	1275186	1539016
2000	64936	2214161	2591840
2500	81014	3429492	3888575
5000	191690	13592371	14292032
10000	345526	52999410	53141571

Таблица 4.2: Время работы алгоритмов сортировки на обратно отсортированных данных

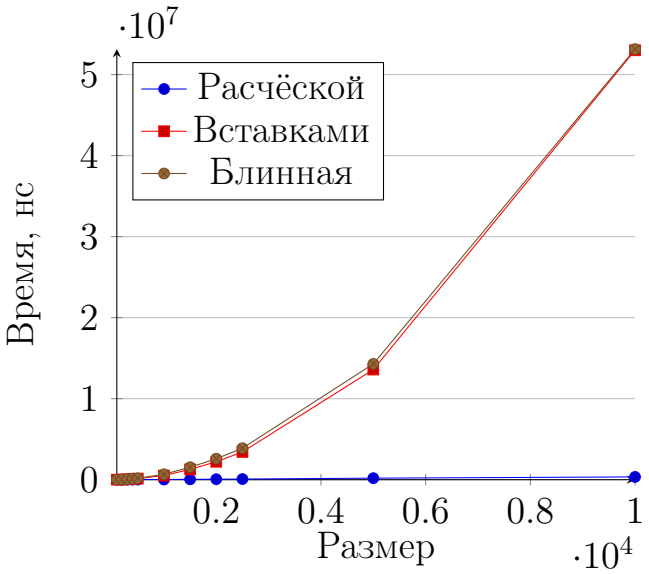

Рис. 4.2: Зависимость времени работы алгоритма сортировки от размера массива, отсортированного в обратном порядке

Размер	Время сортировки, нс		
	Расчёской	**Вставками**	**Блинная**
100	2476	4145	13930
200	7265	15513	51103
300	12514	32334	105035
400	16491	57201	169792
500	22660	80385	230161
1000	73535	360518	1026597
1500	102258	769226	2144729
2000	155006	1331459	3598224
2500	216796	2147710	5948125
5000	500871	7779561	23519061
10000	964756	27076732	88348471

Таблица 4.3: Время работы алгоритмов сортировки на случайных данных

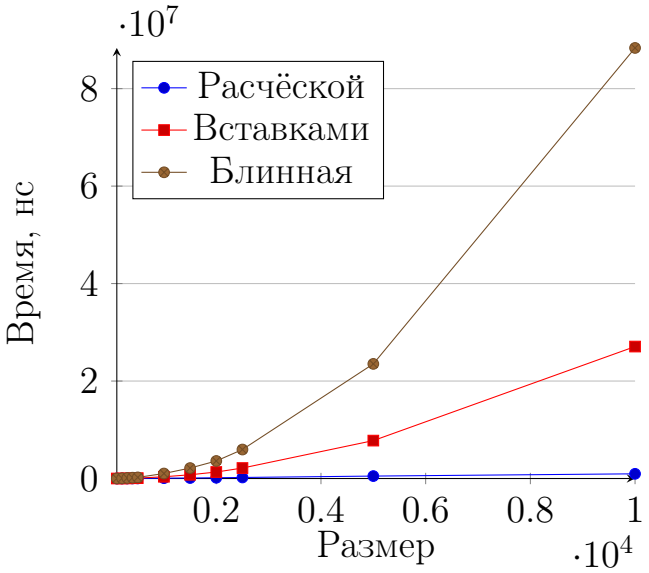

Рис. 4.3: Зависимость времени работы алгоритма сортировки от размера случайного массива

Вывод

Алгоритм сортировки расчёской работает лучше остальных двух на всех случаях расположения элементов в массиве.

Заключение

В рамках лабораторной работы:

- были изучены и реализованы 3 алгоритма сортировки: расчёской, вставками, блинная;
- был проведен сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- был проведен сравнительный анализ алгоритмов на основе экспериментальных данных;
- был подготовлен отчет по проделанной работе.

Наиболее эффективной оказалась сортировка расчёской, выигрывая по времени на случайных данных сортировку вставками и блинную. Так же важно отметить, что сортировка расчёской в худшем случае (случай, когда элементы массива отсортированы в обратном порядке) обгоняет остальные два алгоритма в следствие того, что в алгоритме присутствует флаг, который отмечает, была ли перестановк на итерации внешнего цикла.

Список использованных источников

Список использованных источников

[1] Кнут Дональд. Сортировка и поиск. Вильямс, 2000. Т. 3 из *Искусство программирования*. с. 834.

[2] Язык программирования Go [Электронный ресурс]. Режим доступа: `https://go.dev` (дата обращения: 21.09.2022).

[3] Документация по ЯП Go: бенчмарки [Электронный ресурс]. Режим доступа: `https://go.dev/doc/tutorial/add-a-test` (дата обращения: 21.09.2022).

[4] Windows 11 [Электронный ресурс]. Режим доступа: `https://www.microsoft.com/en-us/windows?wa=wsignin1.0` (дата обращения: 21.09.2022).

[5] Процессор AMD Ryzen™ 5 3550H [Электронный ресурс]. Режим доступа: `https://www.amd.com/ru/products/apu/amd-ryzen-5-3550h` (дата обращения: 21.09.2022).