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»

на тему: «Алгоритмы сортировки»

Студент ИУ7-52Б _____ А. П. Лемешев
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Ю. В. Строганов
(Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата) (И. О. Фамилия)

Москва — 2022 г.

СОДЕРЖАНИЕ

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.

1 Аналитическая часть

1.1 Цели и задачи

Цель работы: получить навыки разработки алгоритмов сортировки на трех примерах.

Задачи лабораторной работы:

1) изучить 3 алгоритма сортировки: расческой, вставками, блинная;
2) реализовать изученные алгоритмы сортировки;
3) провести сравнительный анализ трудоемкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
4) провести сравнительный анализ времени работы алгоритмов для различных размеров входного массива;
5) обосновать полученные результаты.

1.2 Алгоритмы сортировки

1.2.1 Сортировка расческой

Сортировка расческой 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 расческой в том, что этот промежуток может быть гораздо больше, чем единица.

1.2.2 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.

1.2.3 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 переворотов. Процесс можно визуально представить как стопку блинов, которую тасуют путем взятия нескольких блинов сверху и их переворачивания.

Вывод

Были рассмотрены три различных алгоритма: сортировка расческой, сортировка вставками и блинная сортировка.

2 Конструкторская часть

2.1 Разработка алгоритмов

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

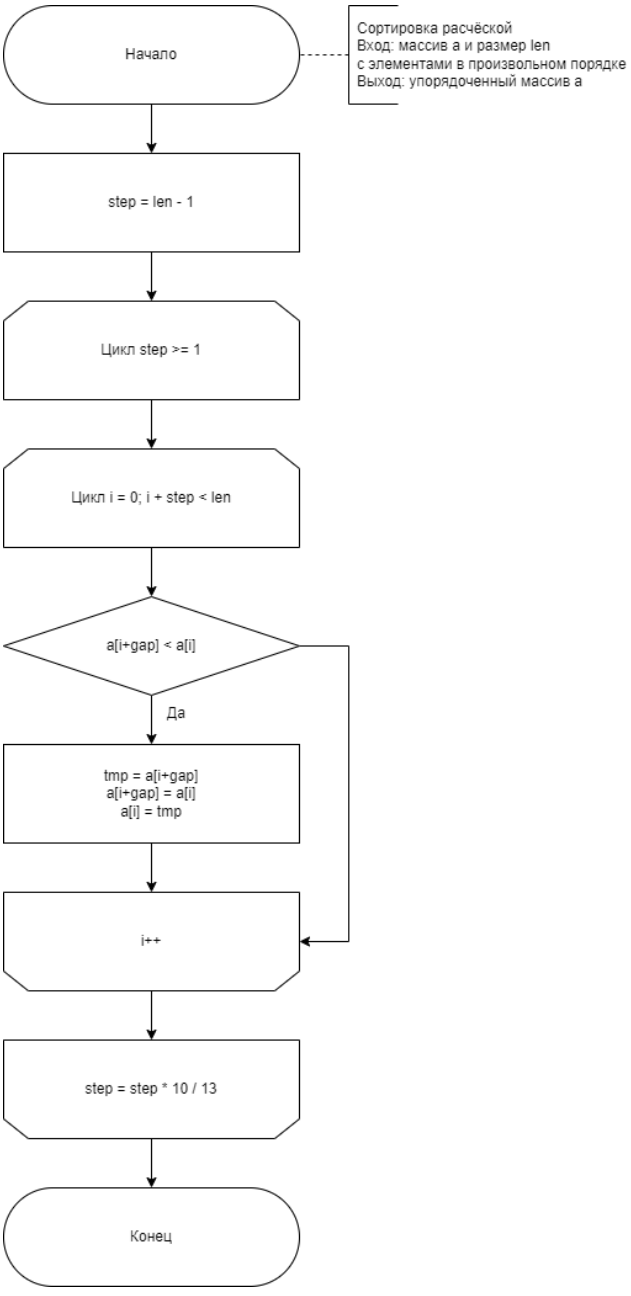

Рисунок 2.1 – Схема алгоритма сортировки расческой

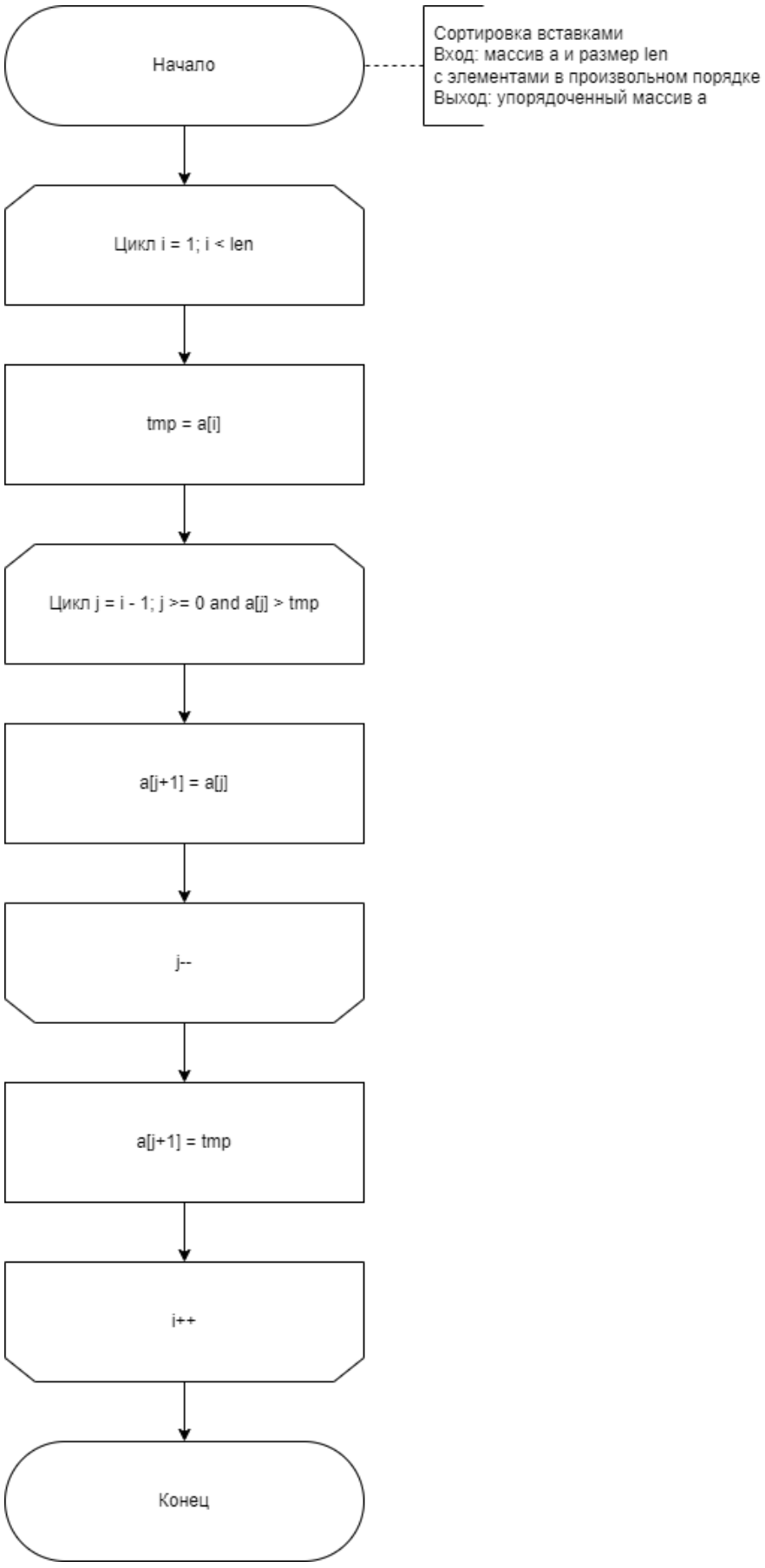

Рисунок 2.2 – Схема алгоритма сортировки вставками

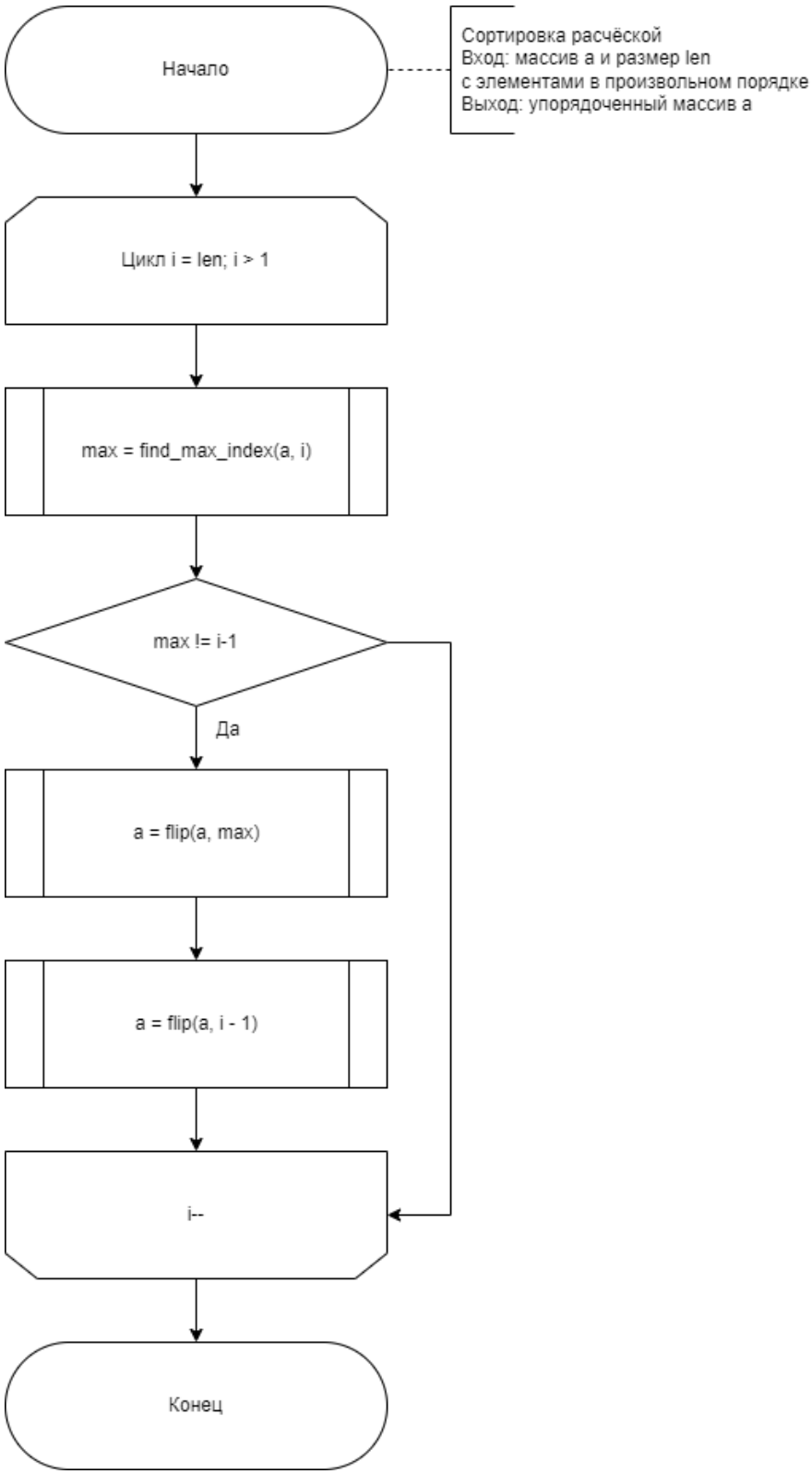

Рисунок 2.3 – Схема алгоритма сортировки блинной

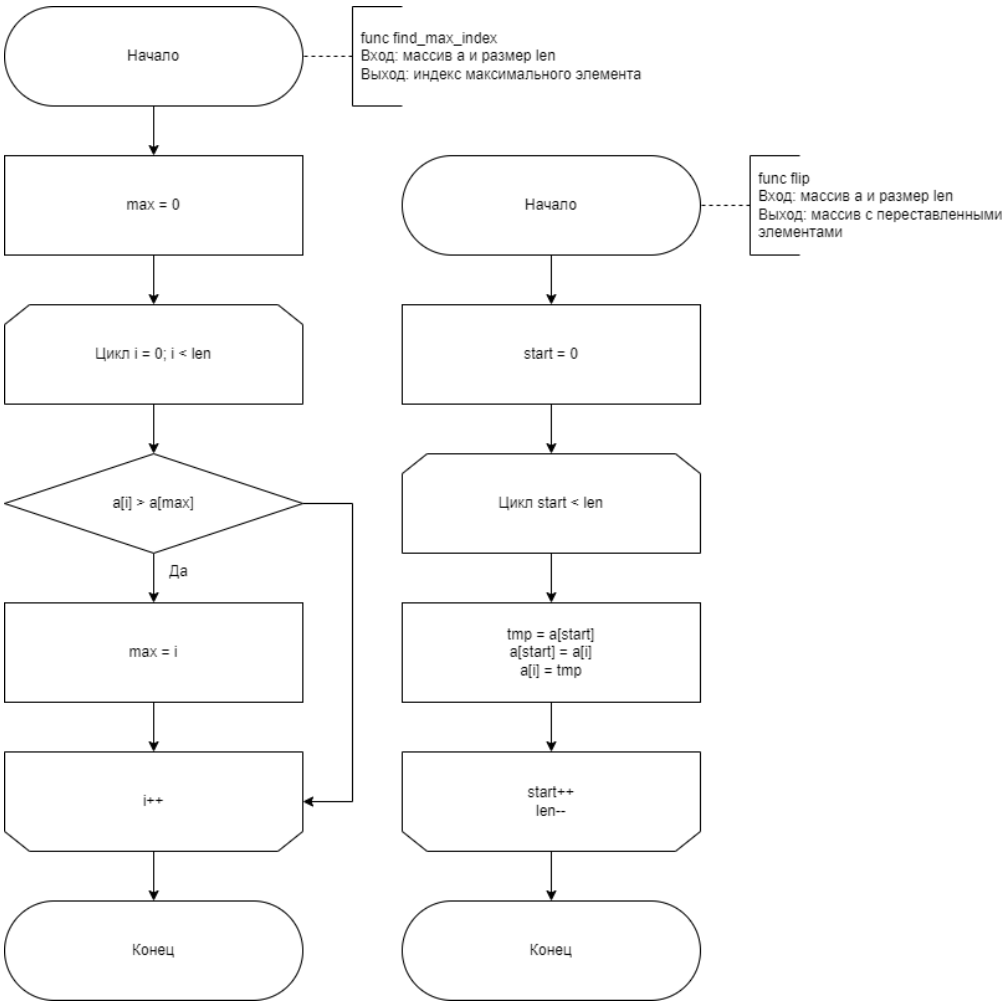

Рисунок 2.4 – Схемы алгоритмов поиска максимального индекса и перестановки элементов в массиве

2.2 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

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

$$+, -, =, + =, - =, ==, ! =, <, >, <=, >=, [], ++, --\qquad (2.1)$$

2) операции из списка (2.2) имеют трудоемкость 2;

$$*, /, \%, * =, / =, \% =\qquad (2.2)$$

3) трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.3);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4) трудоемкость цикла рассчитывается, как (2.4);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.4}$$

5) трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

Обозначим во всех последующих вычислениях размер массивов как N.

2.3.1 Алгоритм сортировки расческой

Рассмотрим трудоемкость реализации алгоритма сортировки расческой.

Трудоемкость сортировки расческой состоит из:

1) предварительных расчетов с трудоемкостью $f(N) = 1 + 2$;

2) двойного цикла, трудоемкость которого равна $f(N) = \underbrace{1}_{\text{срав}} + log_t(N) \cdot (\underbrace{1}_{f_{\text{иниц}}} + \underbrace{2}_{f_{\text{срав}}} + 9N + \underbrace{1}_{f_{\text{инк}}}) + \underbrace{2}_{f_{\text{деления}}}$,

где t — фактор, в нашем случае 1.247.

Трудоемкость при лучшем случае (отсортированный массив): $f(N) = log_t(N) \cdot (4 + N) + 3$.

Однако стоит учитывать, что при факторе ≈ 1.3, трудоемкость аппроксимируется след. образом: $log_t(N) \cdot (4 + N) + 3 \approx C \cdot log(N) \cdot N \approx O(N \cdot log(N))$, где C — некая константа.

В работе [2] сообщается о абстрактной аппроксимации и просьбе быть «оптимистичными». Наличие в рассуждении «просьбы» является минусом, так как читателю придется взять на «веру» сказанное. В работе [3] ведется рассуждение о выводе худшего случая экспериментальным путем,

однако стоит сказать, что экспериментальный путь не является гарантией полученной асимптотики. В работе [4] вводится математический расчет нижней границы худшего случая. В данной работе приводится огромное количество аппроксимаций. В работе [5] ведется разговор о приближении асимптотики худшего случая. В работе [6] приводится матемаческий расчет асимпотики среднего случая, указание о том, что средний случай рассчитать сложнее, чем худший. Доказательство асимптотики среднего и худшего случая.

Стоит обратить внимание, что в перечисленных работах происходит расчет худшего случая, но не приводятся конкретные примеры.

Трудоемкость при худшем случае: $f(N) = log_t(N) \cdot (4 + 9N) + 3 \approx O(N^2)$.

2.3.2 Алгоритм сортировки вставками

Рассмотрим трудоемкость реализации алгоритма сортировки вставками.

Алгоритм сортировки вставками состоит только из двойного цикла, соответственно $f_{\text{вставок}} = f_{\text{цикла с вложенностью 2}}$.

Рассчитаем трудоемкость для вложенного цикла: $f_{\text{цикла с вложенностью 2}} = 1 + 1 + N \cdot (1 + 6 + \frac{N+1}{2} \cdot 8 + 1) + 1$.

Трудоемкость при лучшем случае (отсортированный массив). Алгоритм ни разу не войдет во внутренний цикл: $3 + N \cdot (\frac{N+1}{2} \cdot 0 + 8) = 3 + 8N \approx O(N)$

Трудоемкость при худшем случае (отсортированный в обратном порядке массив). Алгоритм будет полностью проходить внутренний цикл каждый раз: $3 + N \cdot (8 + \frac{N+1}{2} \cdot 8) = 3 + 8N + 8N \cdot \frac{N+1}{2} \cdot 8 \approx O(N^2)$.

2.3.3 Алгоритм блинной сортировки

Рассмотрим трудоемкость реализации алгоритма блинной сортировки.

Этот алгоритм сортировки состоит только из двойного цикла, соответственно $f_{\text{блинная}} = f_{\text{цикла с вложенностью 2}}$.

Рассчитаем трудоемкость для вложенного цикла: $f_{\text{цикла с вложенностью 2}} = 1 + 1 + N \cdot (3 + 5 \cdot N + 1 + 1 + 4 \cdot 2 \cdot N) + 1$.

Трудоемкость при лучшем случае (алгоритм ни разу не сделает

переворот): $3 + N \cdot (\frac{N+1}{2} \cdot 0 + 8) = 3 + 4N + 5N^2 \approx O(N^2)$

Трудоемкость при худшем случае (отсортированный в обратном порядке массив). Алгоритм будет полностью проходить внутренний цикл каждый раз: $3 + N \cdot (8 + \frac{N+1}{2} \cdot 8) = 3 + 9N + 13N^2 \approx O(N^2)$.

Стоит отметить, что асимптотики худшего и лучшего случаев совпадает.

Вывод

На основе формул и теоретических данных, полученных в аналитическом разделе, были спроектированы схемы алгоритмов. Для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

3 Технологическая часть

3.1 Требования к к программному обеспечению

К программе предъявляется ряд требований:

— на вход подается массив сравнимых элементов;

— программа должна измерять реальное время;

— 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 [7]. 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 [8].

3.3 Листинг кода

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

Листинг 3.1 – Алгоритм сортировки расческой

```
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, repeats int) {
	var durations [3]uint64
	var tmp array.Array

	for i := 0; i < repeats; i++ {
		tmp = arr.Copy()
		start := C.tick()
		tmp.SortComb()
		durations[0] += uint64(C.tick() - start)

		tmp = arr.Copy()
		start = C.tick()
		tmp.SortInsert()
		durations[1] += uint64(C.tick() - start)

		tmp = arr.Copy()
		start = C.tick()
		tmp.SortPancake()
		durations[2] += uint64(C.tick() - start)
	}
}
```

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

На основе схем из конструкторского раздела были разработаны и протестированы спроектированные алгоритмы.

4 Исследовательская часть

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими техническими характеристиками.

1. Операционная система: Windows 11 x64 [9].
2. Память: 8 GiB.
3. Процессор: AMD Ryzen 5 3550H [10].

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		
	Расческой	**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

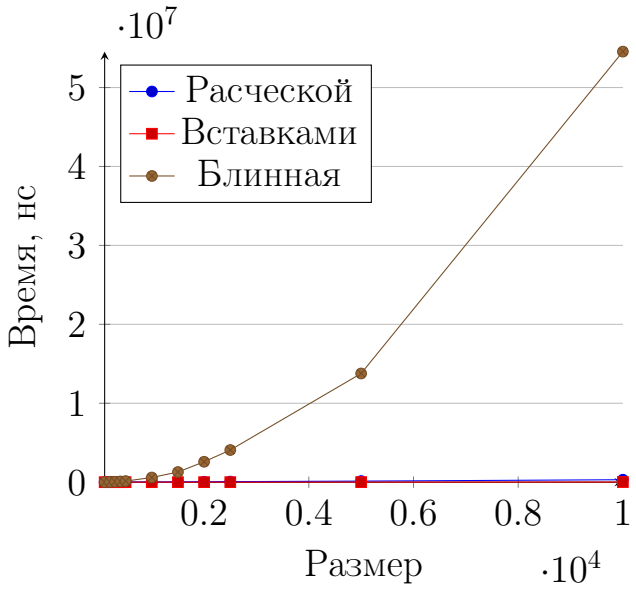

Рисунок 4.1 – Зависимость времени работы алгоритма сортировки от размера отсортированного массива

Размер	Время сортировки, нс		
	Расческой	**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

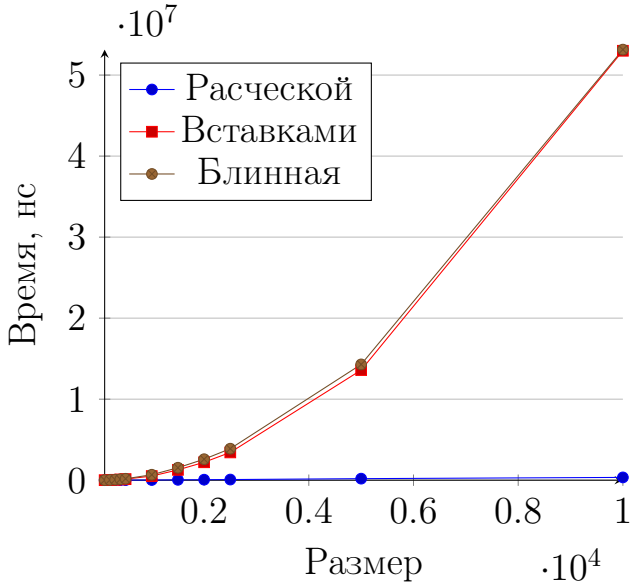

Рисунок 4.2 – Зависимость времени работы алгоритма сортировки от размера массива, отсортированного в обратном порядке

Размер	Время сортировки, нс		
	Расческой	**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

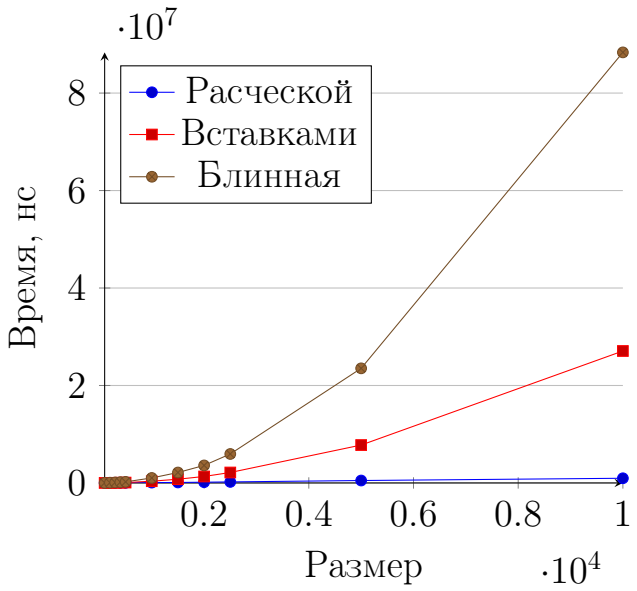

Рисунок 4.3 – Зависимость времени работы алгоритма сортировки от размера случайного массива

Вывод

В данном разделе были сравнены реализованные алгоритмы по времени. Алгоритм сортировки расческой работает лучше остальных при случайном расположении элементов в массиве.

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы была достигнута поставленная цель: были изучены принципы разработки алгоритмов сортировки на трех примерах.

Решены все поставленные задачи:

1) изучены 3 алгоритма сортировки: расческой, вставками, блинная;
2) реализованы изученные алгоритмы сортировки;
3) проведен сравнительный анализ трудоемкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
4) проведен сравнительный анализ времени работы алгоритмов для различных размеров входного массива;
5) обоснованы полученные результаты.

Исходя из результатов, полученных в исследовательской части, получилось соотнести время выполнения с трудоемкостью для всех алгоритмов. Для массивов со случайным расположением элементов наиболее эффективной реализацией из рассмотренных являеться алгоритм сортировки расческой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Кнут Д.* Сортировка и поиск. Т. 3. — Вильямс, 2000. — С. 834. — (Искусство программирования).
2. BYTE 1991 04 OCR [Электронный ресурс]. — Режим доступа: https://archive.org/details/eu_BYTE-1991-04_OCR/page/n425/mode/2up (дата обращения: 06.11.2022).
3. A comparative Study of Sorting Algorithms Comb, Cocktail and Counting Sorting [Электронный ресурс]. — Режим доступа: https://www.irjet.net/archives/V4/i1/IRJET-V4I1249.pdf (дата обращения: 06.11.2022).
4. Analysis of Sorting Algorithms by Kolmogorov Complexity [Электронный ресурс]. — Режим доступа: https://arxiv.org/pdf/0905.4452.pdf (дата обращения: 06.11.2022).
5. SIMD – and Cache-Friendly Algorithm for Sorting an Array of Structures [Электронный ресурс]. — Режим доступа: https://www.vldb.org/pvldb/vol8/p1274-inoue.pdf (дата обращения: 06.11.2022).
6. Dobosiewicz Sort and Shakersort [Электронный ресурс]. — Режим доступа: https://homepages.cwi.nl/~paulv/papers/sorting.pdf (дата обращения 8.11.2022).
7. Язык программирования Go [Электронный ресурс]. — Режим доступа: https://go.dev (дата обращения: 21.09.2022).
8. Документация по ЯП Go: бенчмарки [Электронный ресурс]. — Режим доступа: https://go.dev/doc/tutorial/add-a-test (дата обращения: 21.09.2022).
9. Windows 11 [Электронный ресурс]. — Режим доступа: https://www.microsoft.com/en-us/windows?wa=wsignin1.0 (дата обращения: 21.09.2022).
10. 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).