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

Одной из важнейших процедур обработки структурированной информации является сортировка. Сортировкой называют процесс перегруппировки заданной последовательности (кортежа) объектов в некотором определенном порядке. Определенный порядок (например, упорядочение в алфавитном порядке, по возрастанию или убыванию количественных характеристик, по классам, типам и.т.п.) в последовательности объектов необходимо для удобства работы с этим объектом . В частности, одной из целей сортировки является облегчение последующего поиска элементов в отсортированном множестве.

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.

Как уже было сказано, в любой сфере, использующей какое-либо программное обеспечение, с большой долей вероятности используются сорти-

ровки. К примеру, на сайте можно найти результаты производительности алгоритмов сортировки для ряда ведущих центров данных. При этом используются различные критерии оценки эффективности.

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 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

2.1 Требования к ПО

К программе предъявляется ряд требований:

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

2.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран современный компилируемый ЯП Golang [?]. 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 [?].

2.3 Листинг кода

В листингах 2.1 – 2.3 приведены листинги алгоритма сортировки расчёской, вставками и блинной соответственно. В листингах 2.4 – 2.5 приведены примеры реализации тестов и бенчмарков.

Листинг 2.1: Алгоритм сортировки расчёской

```
func (a Array) SortComb() {
	gap := a.Len()
	flag := true

	for gap != 1 || flag {
		nextGap(&gap)
		flag = false

		for i := 0; i < a.Len() - gap; i++ {
			if a.Less(i + gap, i) {
				a.Swap(i + gap, i)
				flag = true
			}
		}
	}
}

func nextGap(gap *int) {
	*gap = (*gap * 10) / 13
	if *gap < 1 { *gap = 1 }
}
```

Листинг 2.2: 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 2.3: 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 2.4: Пример реализации теста

```
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 2.5: Пример реализации бенчмарка

```
func benchmark(arr array.Array) error {
	durations := [3]time.Duration{}
	start := time.Time{}
	repeats := *repeatsFlag

	var arr2, arr3, arr4 array.Array

	for i := 0; i < repeats; i++ {
		arr2 = arr.Copy()
		arr3 = arr.Copy()
		arr4 = arr.Copy()

		start = time.Now()
		arr2.SortComb()
		durations[0] += time.Since(start)

		start = time.Now()
		arr3.SortInsert()
		durations[1] += time.Since(start)

		start = time.Now()
		arr4.SortPancake()
		durations[2] += time.Since(start)
	}

    return nil
}
```

2.4 Тестирование функций

В таблице 2.1 приведены тесты для функций, реализующих алгоритмы сортировки. Тесты пройдены успешно.

Таблица 2.1: 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.