Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курский государственный университет»

Кафедра программного обеспечения и администрирования информационных систем

Направление подготовки математическое обеспечение и администрирование информационных систем

Форма обучения очная

Отчет
по лабораторной работе №1
«Сравнительный анализ методов сортировки»

Выполнил:
студент группы 213 Файтельсон А.А.

Проверил:
ассистент кафедры ПОиАИС Овсянников А.В.

Курск, 2024

Цель работы: изучение методов сортировки массивов и приобретение навыков в проведении сравнительного анализа различных методов сортировки.

Постановка задачи:

1. Изучить временные характеристики алгоритмов.
2. Изучить методы сортировки.
3. Программно реализовать 3 метода сортировки массивов в соответствии с вариантом:

Таблица 1 – Индивидуальный вариант

Вариант 8	Метод 1.	14. Сортировка методом естественного двухпутевого слияния
	Метод 2.	5. Сортировка Шелла
	Метод 3.	8. Обменная сортировка

1. Разработать и программно реализовать средство для проведения экспериментов по определению временных характеристик алгоритмов сортировки.
2. Провести эксперименты по определению временных характеристик алгоритмов сортировки. Результаты экспериментов представить в виде таблиц 1, 2 и 3, клетки которых содержат время выполнения алгоритма сортировки массива с заданным количеством элементов.
3. Построить график зависимости количества операций сравнения от количества элементов в сортируемом массиве.
4. Определить аналитическое выражение функции зависимости количества операций сравнения от количества элементов в массиве.
5. Определить порядок функций временной сложности алгоритмов сортировки при сортировке упорядоченных, неупорядоченных и упорядоченных в обратном порядке массивов.

Алгоритмы решения задачи в текстуальном виде.

14. Сортировка методом естественного двухпутевого слияния.

В основе сортировки таблицы методом слияния лежит процедура слияния двух упорядоченных таблиц. Эти таблицы должны быть объединены таким образом, чтобы получилась одна упорядоченная таблица.

Сущность метода естественного двухпутевого слияния состоит в том, что упорядочиваемая таблица на каждом шаге сортировки представляется равными упорядоченными группами элементов, которые попарно сливаются, образуя новые группы, содержащие вдвое больше элементов (на первом шаге группы состоят из одного элемента). От шага к шагу количество групп уменьшается вдвое, пока не будет получена одна упорядоченная группа.

5. Сортировка Шелла.

Сортировка Шелла — это улучшенный алгоритм сортировки, основанный на методе вставками. В отличие от обычной сортировки вставками, в которой элементы перемещаются на одну позицию, сортировка Шелла сравнивает и перемещает элементы, находящиеся на определённом расстоянии друг от друга (так называемый "шаг"). Изначально шаг может быть большим, и по мере выполнения алгоритма он уменьшается, приближаясь к 1. Таким образом, крупные элементы быстро перемещаются к своей позиции, а для маленьких шагов сортировка завершается более эффективно. Это ускоряет процесс по сравнению с классической сортировкой вставками.

8. Обменная сортировка.

Обменная сортировка (схема Батчера) — это алгоритм сортировки, в котором элементы массива сравниваются попарно и при необходимости меняются местами, чтобы привести их в правильный порядок. Процесс делится на несколько этапов, где элементы сравниваются и перемещаются по заранее определенной схеме. Этот метод сортировки гарантирует, что все элементы окажутся на своих местах в конце выполнения алгоритма. Особенностью схемы Батчера является её параллельность

— многие сравнения могут происходить одновременно, что делает этот алгоритм особенно эффективным для аппаратной реализации.

Результаты сравнительного анализа алгоритмов.
(в микросекундах)

Таблица 2 – Замеры для упорядоченного массива

Сортировка	Количество элементов в массиве		
	5	10	15
5.	0.3	0.2	0.25
8.	0.3	0.45	0.55
14.	1.7	2.6	3.5

Количество элементов в массиве		
20	25	30
0.3	0.36	0.47
0.84	0.85	0.88
4.8	4.7	5

Количество элементов в массиве		
35	40	45
0.41	0.34	0.38
1.1	1.35	1.3
6.4	6.6	7.2

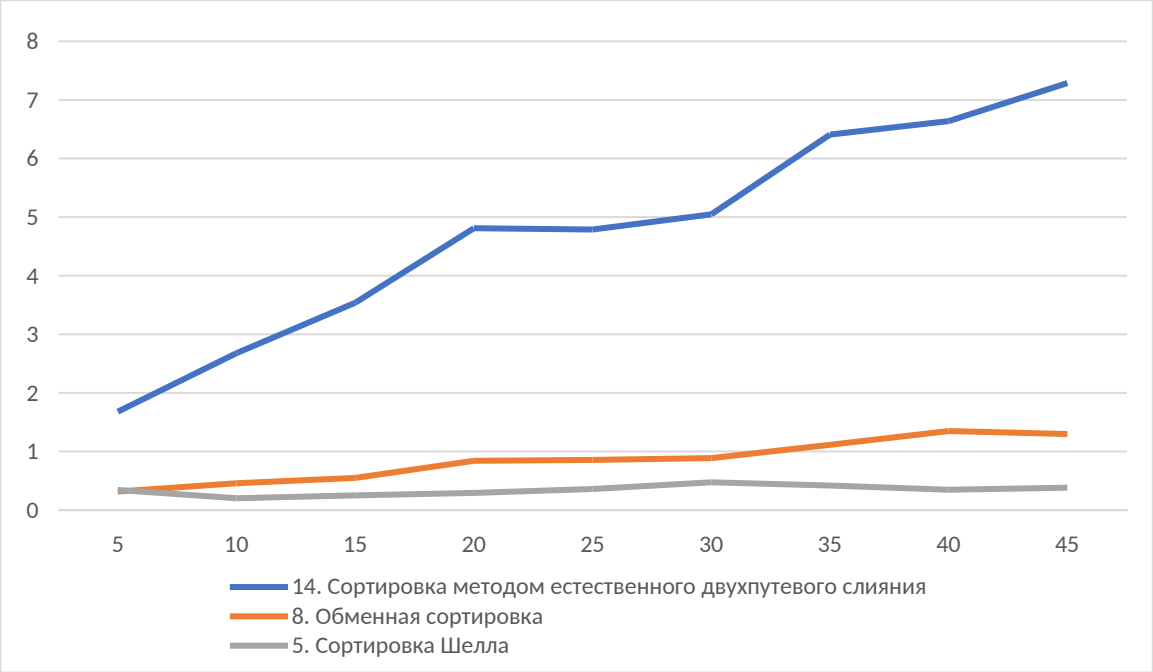

Рисунок 1 – Зависимость затраченного на сортировку времени от количества элементов в упорядоченном массиве

Таблица 3 – Замеры для массива, упорядоченный в обратном порядке

Сортировка	Количество элементов в массиве		
	5	10	15
5.	0.14	0.21	0.42
8.	0.29	0.495	0.64
14.	1.4	2.6	4

Количество элементов в массиве		
20	25	30
0.52	0.59	0.96
0.98	1.2	1.4
3.9	4.7	10

Количество элементов в массиве		
35	40	45
1.09	1.05	1.082
1.9	2.2	2.3
6.4	32.5	7.8

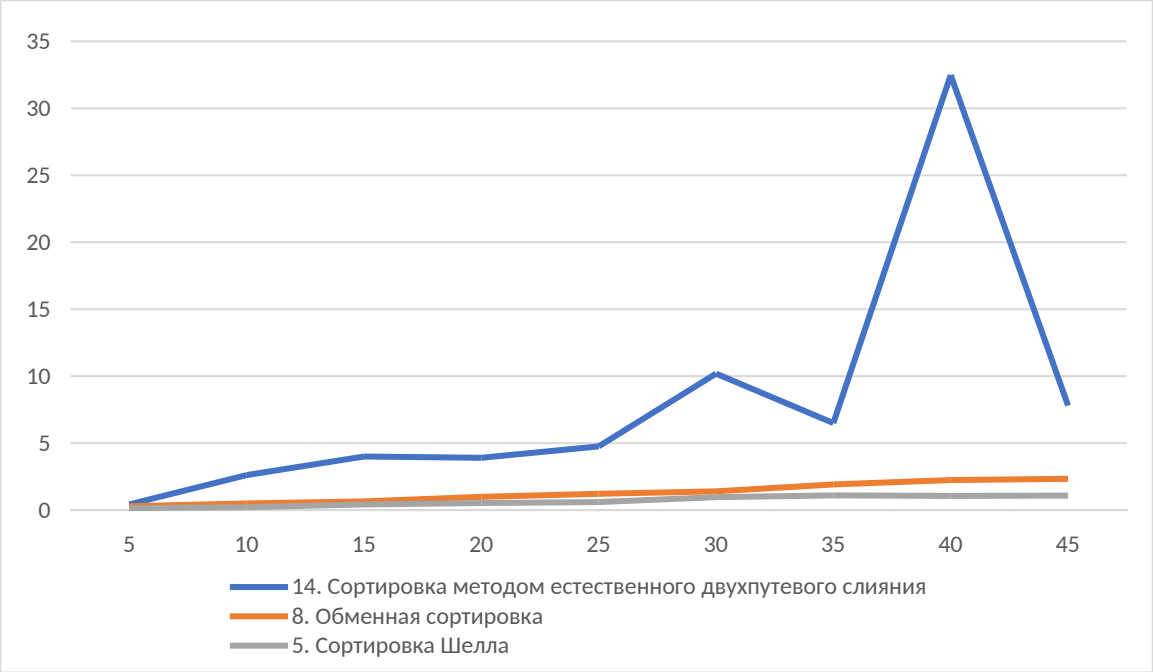

Рисунок 2 – Зависимость затраченного на сортировку времени от количества элементов в упорядоченном в обратном порядке массиве

Таблица 4 – Замеры для неупорядоченного массива

Сортировка	Количество элементов в массиве		
	5	10	15
5.	0.16	0.24	0.52
8.	0.27	0.54	0.71
14.	1.2	6.6	3.5

Количество элементов в массиве		
20	25	30
0.61	1.05	1.17
1.04	1.4	1.76
4.44	5	5.6

Количество элементов в массиве		
35	40	45
1.57	1.9	2.02
2.04	2.41	2.8
6.8	13	9

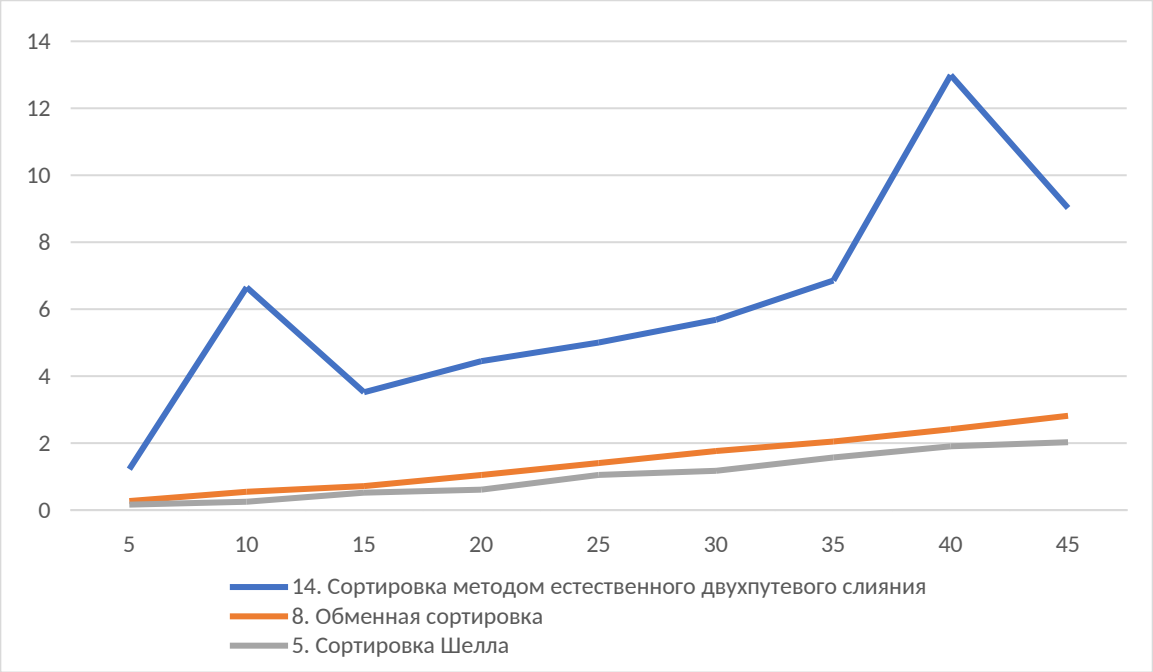

Рисунок 3 – Зависимость затраченного на сортировку времени от количества элементов в неупорядоченном массиве

Листинг программы

```
package main

import (
	"fmt"
	"sort"
	"time"
)

func shellSort(arr []int) []int {
	for gap := len(arr) / 2; gap > 0; gap /= 2 {
		for i := gap; i < len(arr); i++ {
			// insertion sort with a gap
			for j := i; j >= gap && arr[j-gap] > arr[j]; j -= gap {
				arr[j], arr[j-gap] = arr[j-gap], arr[j]
			}
		}
	}
	return arr
}

func FloorLog2(value int) int {
	result := -1
	for i := 1; i < value; i <<= 1 {
		result++
```

```
	}
	return result
}

func BatcherSort(arr []int) []int {
	array := arr
	length := len(array)
	t := FloorLog2(length)
	p0 := (1 << t)
	p := p0
	for {
		q := p0
		r := 0
		d := p
		for ok := true; ok; ok = (r == 0 || q != p) {
			if r != 0 {
				d = q - p
				q >>= 1
			}
			for i := 0; i < length-d; i++ {
				if ((i & p) == r) && array[i] > array[i+d] {
					array[i], array[i+d] = array[i+d], array[i]
				}
			}
			r = p
		}
		p >>= 1
		if p <= 0 {
			break
		}
	}
	return array
}

func merge(a []int, b []int) []int {
	r := make([]int, 0)
	i := 0
	j := 0
	// 注意此时的a, b都是已经排好序的
	//  每次都是a[i],  b[j]比较，小的先放到r里面，此时"小的所在的数组"i加一位继续与b[j]对比
	for i < len(a) && j < len(b) {
		if a[i] <= b[j] {
			r = append(r, a[i])
			i++
```

```
            } else {
                r = append(r, b[j])
                j++
            }
        }
        // 上面for循环结束后， 要么a还有数据， 要么b还有数据，即下面两个for循环只有一个会运行到
        // 因为剩下的数据是已经排好序的， 所有直接插入就可以
        for i < len(a) {
            r = append(r, a[i])
            i++
        }
        for j < len(b) {
            r = append(r, b[j])
            j++
        }
        return r
}

func MergeSort(arr []int) []int {
        if len(arr) < 2 {
            return arr
        }
        middle := len(arr) / 2
        a := MergeSort(arr[:middle])
        b := MergeSort(arr[middle:])
        return merge(a, b)
}

func getDuration(arr []int, f func(arr []int) []int) {
        before := make([]int, len(arr))
        copy(before, arr)
        start := time.Now()
        after := f(before)
        _ = after // do nothing
        duration := time.Since(start)
        fmt.Println("время сортировки - ", duration)
        // fmt.Println(after)
}

func printArrs(arrs [][]int) {
        fmt.Println("bebra _____ bebra _____ ")
        fmt.Println(arrs)
}
```

```
func giveArray() [][]int {
	arrs := [][]int{
		{45, 47, 85, 78, 3},
		{36, 22, 16, 78, 49, 31, 57, 79, 6, 18},
		{1, 53, 43, 4, 2, 60, 37, 26, 97, 40, 65, 81, 93, 51, 2},
		{17, 35, 10, 66, 13, 74, 36, 77, 3, 94, 35, 24, 57, 90, 97, 12, 73, 61, 97, 56},
		{40, 19, 90, 21, 72, 76, 3, 90, 2, 67, 12, 77, 27, 63, 24, 48, 47, 58, 31, 9, 86,
74, 97, 39, 73},
		{74, 17, 68, 23, 27, 46, 34, 3, 97, 6, 8, 9, 92, 1, 23, 34, 88, 1, 28, 100, 88, 81,
6, 21, 23, 13, 23, 30, 8, 66},
		{12, 31, 29, 65, 53, 58, 65, 56, 47, 44, 75, 21, 71, 43, 95, 17, 59, 2, 6, 36, 24,
40, 24, 14, 16, 17, 17, 72, 6, 7, 3, 97, 28, 15, 48},
		{67, 70, 4, 75, 68, 58, 60, 96, 2, 37, 34, 13, 17, 11, 66, 9, 58, 58, 53, 71, 53,
53, 35, 86, 73, 6, 18, 43, 52, 51, 9, 36, 99, 17, 84, 40, 91, 85, 47, 73},
		{4, 76, 25, 60, 42, 79, 37, 9, 5, 59, 94, 36, 18, 70, 82, 19, 30, 90, 49, 78, 29,
91, 52, 85, 95, 18, 93, 25, 82, 7, 61, 85, 44, 14, 94, 61, 81, 3, 45, 41, 78, 99, 48, 63, 23},
	}
	return arrs
}

func giveReverseArray(arr [][]int) [][]int {
	array := make([][]int, len(arr))
	for i := range arr {
		array[i] = make([]int, len(arr[i]))
		copy(array[i], arr[i])
	}

	for g := 0; g < len(array); g++ {
		for i, j := 0, len(array[g])-1; i < j; i, j = i+1, j-1 {
			array[g][i], array[g][j] = array[g][j], array[g][i]
		}
	}
	return array
}

func giveSortedArray(arr [][]int) [][]int {
	array := make([][]int, len(arr))
	for i := range arr {
		array[i] = make([]int, len(arr[i]))
		copy(array[i], arr[i])
		sort.Ints(array[i])
	}
	return array
}
```

```
func testing(arrs [][]int, debug bool) {
	// Shell testing
	fmt.Println("Shell")
	for i := 0; i < len(arrs); i++ {
		getDuration(arrs[i], shellSort)
	}
	if debug {
		printArrs(arrs)
	}
	// Butcher testing
	fmt.Println("Butcher")
	for i := 0; i < len(arrs); i++ {
		getDuration(arrs[i], BatcherSort)
	}

	if debug {
		printArrs(arrs)
	}
	// Butcher testing
	// merge testing
	fmt.Println("Merge")
	for i := 0; i < len(arrs); i++ {
		getDuration(arrs[i], MergeSort)
	}
	if debug {
		printArrs(arrs)
	}
}

func main() {
	// arr := []int{12, 34, 54, 2, 3, 1, 11, 8, 7}
	// fmt.Println(arr)
	// getDuration(arr, BatcherSort)
	// arr5 := []int{64, 34, 25, 12, 22}
	// arr10 := []int{64, 34, 25, 12, 22, 11, 90, 70, 1, 0}
	// arr15 := []int{64, 34, 25, 12, 22, 11, 90, 70, 1, 0, 64, 34, 25, 12, 22}
	// arr20 := []int{64, 34, 25, 12, 22, 11, 90, 70, 1, 0, 64, 34, 25, 12, 22, 64, 34, 25, 12,
22}
	// arr25 := []int{64, 34, 25, 12, 22, 11, 90, 70, 1, 0, 64, 34, 25, 12, 22, 64, 34, 25, 12,
22, 64, 34, 25, 12, 22}
	// arr30 := []int{64, 34, 25, 12, 22, 11, 90, 70, 1, 0, 64, 34, 25, 12, 22, 64, 34, 25, 12,
22, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22}
	// arr35 := []int{64, 34, 25, 12, 22, 11, 90, 70, 1, 0, 64, 34, 25, 12, 22, 64, 34, 25, 12,
22, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22}
```

```
	// arr40 := []int{64, 34, 25, 12, 22, 11, 90, 70, 1, 0, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22}
	// arr45 := []int{64, 34, 25, 12, 22, 11, 90, 70, 1, 0, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22, 64, 34, 25, 12, 22}
	// arrs := [][]int{arr5, arr10, arr15, arr20, arr25, arr30, arr35, arr40, arr45}
	arrs := giveArray()
	arrsSorted := giveSortedArray(arrs)
	arrsReversed := giveReverseArray(arrsSorted)

	fmt.Println("Sorted###############################################")
	testing(arrsSorted, false)
	fmt.Println("ReverseSorted----------------------------------")
	testing(arrsReversed, false)
	fmt.Println("Unsorted_____________________________________________")
	testing(arrs, false)
	// fmt.Println("bebra")
	// fmt.Println(arrs)

	// getDuration(arr, BatcherSort)
	// fmt.Println("bebra")
	// fmt.Println(arrs)

	// printArrs(arrs)
	// Butcher testing
}
```