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: «Трудоемкость сортировок»

Студент группы ИУ7-51Б _____ (Подпись, дата) Савинова М. Г. (Фамилия И.О.)

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

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

Москва — 2023 г.

Содержание

Введение

Алгоритмы сортировки — это процессы упорядочивания элементов в определенном порядке.

Общие этапы алгоритмов сортировки включают:

1) *сравнение*: сравнение пар элементов для определения их относительного порядка;

2) *обмен или перемещение*: в зависимости от результата сравнения, элементы могут быть обменены местами или перемещены для достижения необходимого порядка.

3) *повторение*: этапы сравнения и обмена/перемещения повторяются, пока все элементы не будут упорядочены.

В зависимости от конкретного алгоритма сортировки, могут существовать дополнительные этапы, такие как выбор опорного элемента (например, в быстрой сортировке), разделение массива на подмассивы (например, в сортировке слиянием), и другие действия, которые определяются спецификой конкретного алгоритма [1].

Целью данной лабораторной работы является исследование трудоемкостей алгоритмов сортировки.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

1) описать следующие алгоритмы сортировки:
   - поразрядная;
   - расческой;
   - Шелла;
2) релизовать описанные алгоритмы;
3) дать оценку трудоемкости алгоритмов;
4) дать оценку потребляемой памяти реализациями алгоритмов;
5) провести замеры времени выполнения алгоритмов.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы поразраядной сортировки, расческой и Шелла.

1.1 Поразрядная сортировка

Смысл данной сортировки в том, что данные сначала делятся по разрядам и затем сортируются внутри каждого разряда. Алгоритм состоит из нескольких этапов:

1) алгоритм инициализирует индекс рассматриваемого разряда в числах;

2) после чего получает значение данного разряда каждого из чисел с помощью остатка деления на основание системы счисления;

3) затем полученные цифры сортируются;

4) элементы расставляются в соответствии со своими цифрами.

Данный алгоритм повторяется пока индекс рассматриваемого разряда не будет больше числа всех разрядов в числе [1, 2].

1.2 Сортировка расческой

Основная идея «расчёски» в том, чтобы первоначально брать достаточно большое расстояние между сравниваемыми элементами и по мере упорядочивания массива сужать это расстояние вплоть до минимального. Таким образом, мы как бы причёсываем массив, постепенно разглаживая на всё более аккуратные пряди.

Первоначальный разрыв между сравниваемыми элементами лучше брать с учётом специальной величины, называемой фактором уменьшения, оптимальное значение которой равно примерно 1,247. Сначала расстояние между элементами максимально, то есть равно размеру массива

минус один. Затем, пройдя массив с этим шагом, необходимо поделить шаг на фактор уменьшения и пройти по списку вновь. Так продолжается до тех пор, пока разность индексов не достигнет единицы. В этом случае сравниваются соседние элементы как и в сортировке пузырьком, но такая итерация одна.

Оптимальное значение фактора уменьшения $1,24733\cdots = \frac{1}{1-e^{-\Phi}}$, где e — основание натурального логарифма, а $\Phi = 1,61803...$ — золотое сечение [2].

1.3 Сортировка Шелла

Метод предложен в 1959 году и назван по имени автора метода Дональда Шелла. Состоит из прямого и обратного хода. Сравниваются и обмениваются не непосредственные соседи, а элементы, отстоящие на заданном расстоянии.

Когда обнаружена перестановка, цепочка вторичных сравнений охватывает те элементы, которые входили в последовательность первичных просмотров. Каждый последующий просмотр производится с уменьшенным шагом, на последнем просмотре шаг должен равняться 1. Можем использоватьследующую процедуру выбора шага. На первом просмотре шаг имеет значение $d = 2^k - 1$, где k выбрано из условия:

$$2^k < n \leq 2^{k+1}.$$

Новый просмотр производится с шагом $d = \frac{(d-1)}{2}$. Сортировка заканчивается при $d = 0$ [3].

Вывод

В данном разделе были рассмотрены алгоритмы поразрядной сортировки, расческой и Шелла.

2 Конструкторская часть

В данном разделе будут реализованы схемы алгоритмов сортировок и будут приведены рассчеты трудоемкостей для этих алгоритмов.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема поразрядной сортировки.

На рисунке 2.2 представлена схема сортировки расческой.

На рисунке 2.3представлена схема сортировки Шелла.

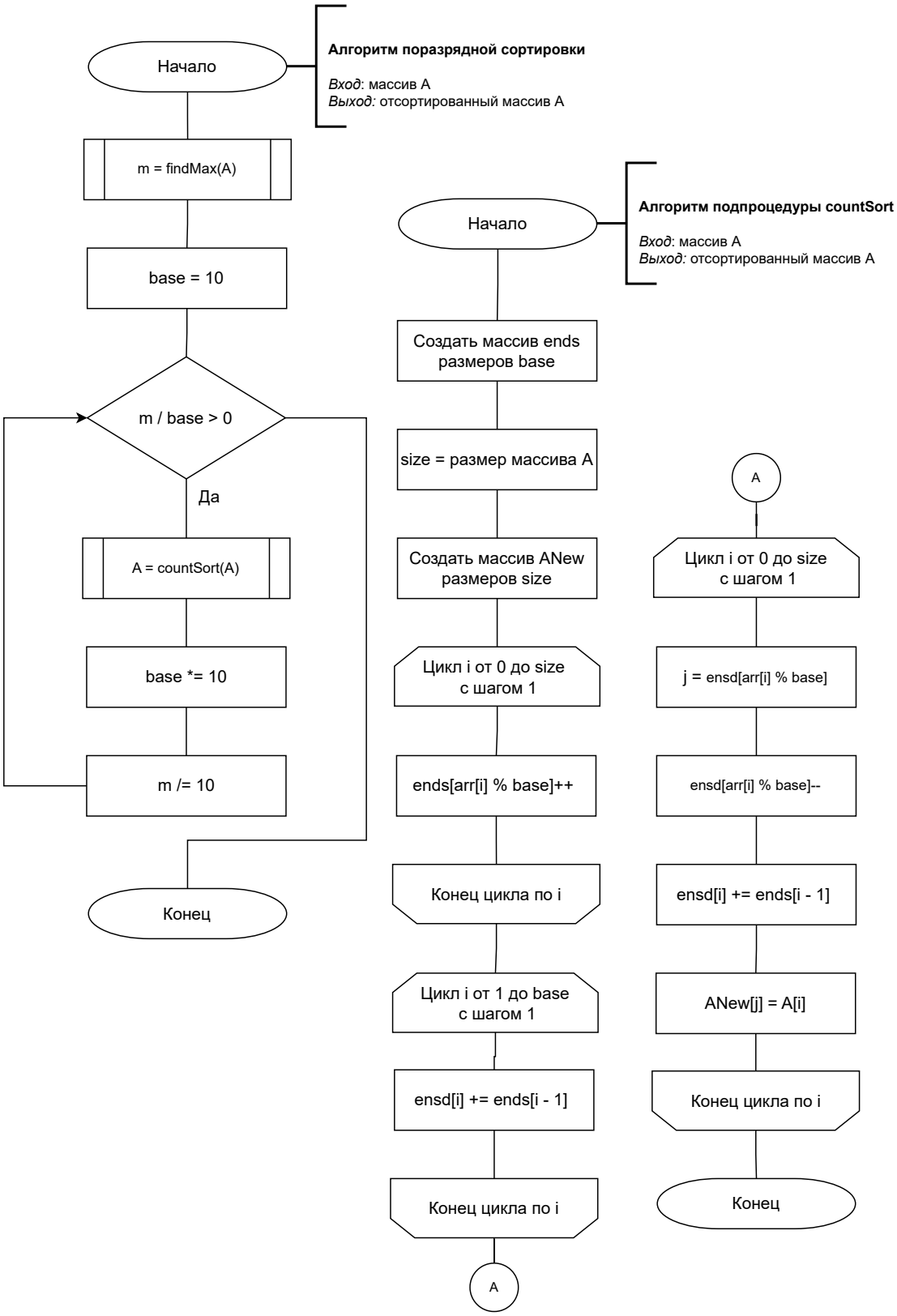

Рисунок 2.1 – Схема алгоритма поразрядной сортировки

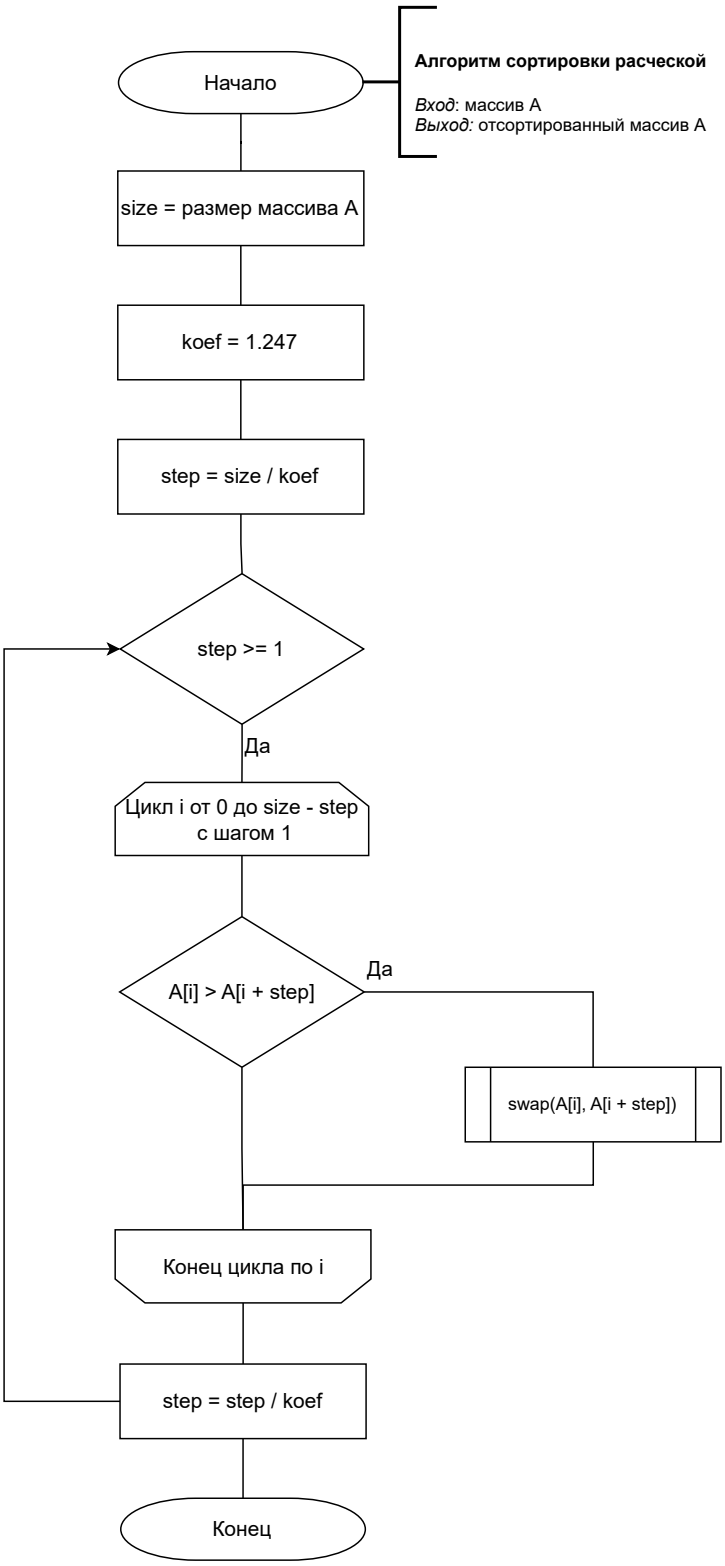

Рисунок 2.2 – Схема алгоритма сортировки расческой

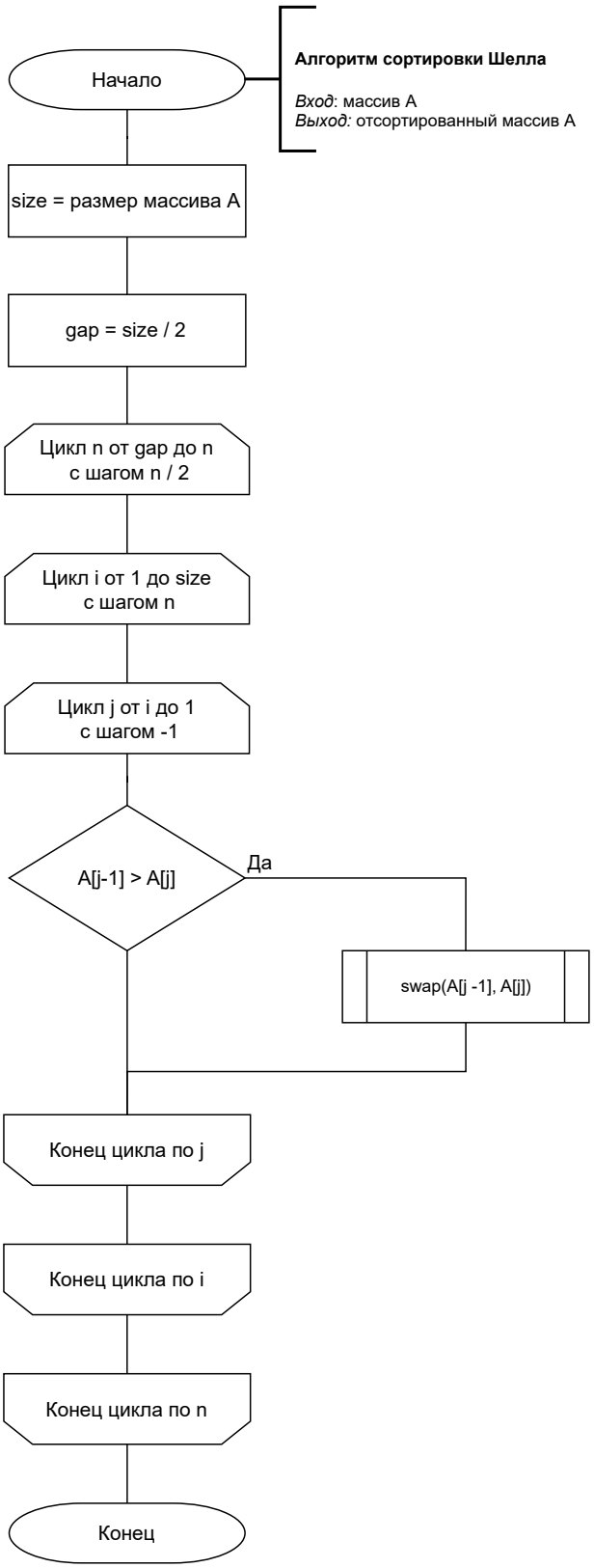

Рисунок 2.3 – Схема алгоритма сортировки Шелла

2.2 Модель вычислений для проведения оценки трудоемкости алгоритмов

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

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

$$+, -, =, + =, - =, ==, ! =, <, >, <=, >=, [], \\ ++, --, \&\&, >>, <<, ||, \&, | \tag{2.1}$$

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

$$*, /, \%, * =, / =, \% = \tag{2.2}$$

3) трудоемкость условного оператора `if условие then A else B` рассчитывается как 2.3;

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{в случае выполнеия условия,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4) трудоемкость цикла рассчитывается как 2.4

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}); \tag{2.4}$$

5) трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

В следующих частях будут приведены рассчеты трудоемкостей алгоритмов сортировок.

Трудоемкость обмена элементов местами равна 5.

Пусть размер сортируемого массива — $size$.

Поразрядная сортировка

Трудоемкость алгоритма поразрядной сортировки равна:

$$f_{radix} = 1 + 1 + m \cdot (1 + 1 + f_{count}). \tag{2.5}$$

где m — число разрядов в максимальном числе; f_{count} — трудоемкость сортировки разрядов подсчетом.

Трудоемкость f_count равна:

$$f_{count} = 1+2+size\cdot(2+4)+2+9\cdot(2+4)+2+size\cdot(2+9) = 17\cdot size+61. \tag{2.6}$$

Итоговая трудоемкость поразрядной сортировки равна:

$$f_{radix} = 2+m\cdot(17\cdot size+61) = 2+17\cdot size\cdot m+61\cdot m \approx O(m\cdot size). \tag{2.7}$$

Сортировка расческой

Трудоемкость алгоритма сортировки расческой равна:

$$f_{comb} = 5 + f_{for_1} \cdot f_{for_2}. \tag{2.8}$$

Трудоемкость внешнего цикла равна:

$$f_{for_1} = 1 + log_{koef} size \cdot (3 + f_{for_2}) \tag{2.9}$$

Трудоемкость внутреннего цикла равна:

Лучший случай: когда элементы массива уже отсортированы или практически (т. е. количество перемещений мало).

$$f_{for_2} = 2 + size \cdot (2 + 4) = 2 + size \cdot 6 \approx O(size). \tag{2.10}$$

Итоговая трудоемкость для лучшего случая равна:

$$f_{comb} = O(size \cdot log_{koef} size). \tag{2.11}$$

Худший случай: когда элементы массива расположены в обратном порядке.

$$f_{for_2} = 2 + size \cdot (2 + 4 + size) = 2 + 2 \cdot size + size^2 \approx O(size^2). \quad (2.12)$$

Итоговая трудоемкость для худшего случая равнв:

$$f_comb = O(size^2). \quad (2.13)$$

Сортировка Шелла

Трудоемкость алгоритма сортировки Шелла равна:

$$f_shell = 3 + f_{for_1} \cdot f_{for_2}. \quad (2.14)$$

Трудоемкость внешнего цикла равна:

$$f_{for_1} = 2 + log_2size\ cdot(2 + f_{for_2}) \approx O(log_2size). \quad (2.15)$$

Трудоемкость внутреннего цикла равна:

$$f_{for_2} = 2 + \frac{size}{n} \cdot (2 + 4 + f_{swap}) \approx O(\frac{size}{n}), \quad (2.16)$$

где n изменяется во внешнем цикле.

Лучший случай: когда элементы массива уже отсортированы или практически (т. е. количество перемещений мало).

$$f_{swap} = O(1), \quad (2.17)$$

Итоговая трудоемкость для лучшего случая равна:

$$f_{shell} = O(log_2size \cdot size). \quad (2.18)$$

Худший случай: когда элементы массива расположены в обратном

порядке.

$$f_{for_3} = 6 + size \cdot (6 + 5 + size) = 6 + 11 \cdot size + size^2 \approx O(size^2). \quad (2.19)$$

Итоговая трудоемкость для худшего случая равнв:

$$f_{shell} = O(size^2). \quad (2.20)$$

Вывод

В данном разделе на основе теоретических данных, полученных в аналитическом разделе, были построены схемы алгоритмов сортировок. Оценены трудоемкости в лучшем и худшем случаях.

Трудоемкость сортировки расческой и Шелла в лучшем случае: $O(size \cdot log_2 size)$; в худшем: $O(size^2)$.

Трудоемкость поразрядной сортировки в большинстве случаев рвна $O(size \cdot m)$.

3 Технологическая часть

В данном разделе будут указаны средства реализации, листинг кода и функциональные тесты.

3.1 Средства реализации

Для реализации данной лабораторной работы был выбран язык `C++` [4], так как в нем есть стандартная библиотека `ctime` [4], которая позволяет производить замеры процессорного времени выполнения программы;

В качестве среды разработки был выбран *Visual Studio Code*: он является кроссплатформенным и предоставляет полный набор интсрументов для проектирования и отладки кода.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — файл содержит точку входа в программу, из которой происходит вызов алгоритмов по разработанному интерфейсу;

— `array.cpp` — файл содержит реализацию класса `ArrayT`;

— `sorts.cpp` — файл содержит функции алгоритомов сортировок;

— `measure.cpp` — файл содержит функции, замеряющие процессорное время выполнения алгоритмов сортировок;

3.3 Реализация алгоритмов

В листинге 3.1 приведена реализация алгоритма поразрядной сортировки.

В листинге 3.2 приведена реализация алгоритма сортировки Шелла.

В листинге 3.3 приведена реализация алгоритма сортировки расческой.

Листинг 3.1 – Функция поразрядной сортировки

```
1  ArrayT Radix::countSort(ArrayT& arr) {
2
3      ArrayT ends(_base, 0);
4
5      int size = arr.size();
6      ArrayT arrNew(size);
7
8      for (int i = 0; i < size; ++i)
9          ends[mod(arr[i], _base)]++;
10
11     for (int i = 1; i < _base; ++i)
12         ends[i] += ends[i - 1];
13
14     for (int i = 0; i < size; ++i) {
15         int& j = ends[mod(arr[i], _base)];
16         --j;
17         arrNew[j] = arr[i];
18     }
19
20     return arrNew;
21 }
22
23 void Radix::execute(ArrayT& arr) {
24
25     int size = arr.size();
26     int maxElem = findMax(arr);
27
28     ArrayT neg, pos;
29
30     for (int i = 0; i < arr.size(); i++) {
31         if (arr[i] < 0)
32             neg.append(arr[i]);
33         else
34             pos.append(arr[i]);
35     }
36
37     _step = 10;
38     _base = _step;
```

```
39
40      while (maxElem) {
41          pos = countSort(pos);
42          neg = countSort(neg);
43
44          _base *= _step;
45
46          maxElem /= _step;
47      }
48
49      arr = neg + pos;
50  }
```

Листинг 3.2 – Функция сортировки Шелла

```
1  void Shell::execute(ArrayT& arr) {
2
3      int size = arr.size();
4      int gap = size / 2;
5
6      for (int n = gap; n > 0; n /= 2) {
7
8          for (int i = n; i < size; i += 1) {
9
10             int j;
11             int tmp = arr[i];
12
13             for (j = i; j >= n && arr[j - n] > tmp; j -= n)
14                 arr[j] = arr[j - n];
15
16             arr[j] = tmp;
17         }
18     }
19 }
```

Листинг 3.3 – Функция сортировки расческой

```
1  void Comb::execute(ArrayT& arr) {
2
3      double koef = 1.247;
4      int size = arr.size();
5
6      int step = static_cast<int>(size / koef);
7
8      while (step >= 1) {
9
10         for (int i = 0; i + step < size; i++)
11
12             if (arr[i] > arr[i + step])
13                 swap(arr[i], arr[i + step]);
14
15         step = static_cast<int>(step / koef);
16     }
17 }
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для разработанных алгоритмов сортировки. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Массив	**Размер**	**Ожидаемый рез.**	**Фактический рез.**		
			Поразрядная	**Шелла**	**Расческой**
1 2 3 4	4	1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4
4 3 2 1	4	4 3 2 1	4 3 2 1	4 3 2 1	4 3 2 1
1 2 3 -9	4	-9 1 2 3	-9 1 2 3	-9 1 2 3	-9 1 2 3
-5 -1 -3 -4 -5	5	-5 -5 -4 -3 -1	-5 -5 -4 -3 -1	-5 -5 -4 -3 -1	

Вывод

Были представлены листинги функций, функциональные тесты. Также в данном разделе была представлена информация о выбранных средствах для разработки алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

— Процессор: AMD Ryzen 5 5500U – 2.10 ГГц;

— Оперативная память: 16 ГБайт;

— Операционная система: Windows 10 Pro 64-разрядная система версии 22H2.

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного ПО.

```
                Меню
1. Radix sort
2. Comb sort;
3. Shell sort

4. Замерить время
5. Замерить память
0. Выход

Выберете пункт (0-4): 1

До: 1000 1 -1 1
После: -1 1 1 1000

                Меню
1. Radix sort
2. Comb sort;
3. Shell sort

4. Замерить время
5. Замерить память
0. Выход

Выберете пункт (0-4): 2

До: 1000 1 -1 1
После: -1 1 1 1000
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Затраты по времени выполнения реализаций алгоритмов

Все замеры проводились на массивах, размером от 1000 до 10000 с шагом 1000. Поскольку замеры по времени имеют некоторую погрешность, замеры производились 100 раз, а затем вычислялось среднее арифметическое значение.

Результаты замеров времени приведены в таблицах 4.1–4.3.

На рисунках 4.2–4.4 приведены графики зависимостей работы алгоритмов от размеров матриц.

Таблица 4.1 – Результаты замеров времени (данные упорядочены по возрастанию)

Размер массива	Время, мкс		
	Поразрядная	Расческой	Шелла
1000	149.98	132.98	83.26
2000	460.36	295.61	170.70
3000	601.04	468.27	286.22
4000	726.33	633.45	367.03
5000	891.82	843.51	516.83
6000	1043.86	1047.36	620.25
7000	1248.68	1284.14	761.96
8000	1352.95	1524.56	828.65
9000	1491.66	1648.02	998.25
10000	1635.23	1885.07	1113.29
11000	3424.52	2061.20	1225.30

Таблица 4.2 – Результаты замеров времени (данные упорядочены по убыванию)

Размер массива	Время, мкс		
	Поразрядная	Расческой	Шелла
1000	305.48	126.75	76.48
2000	441.73	282.94	164.92
3000	611.50	484.92	293.70
4000	776.29	660.69	386.99
5000	908.97	843.72	526.57
6000	1025.28	1017.49	610.98
7000	1172.44	1233.47	718.44
8000	1325.12	1459.71	821.33
9000	1460.68	1631.14	999.42
10000	3225.52	1897.15	1139.28
11000	3389.68	2069.76	1225.05

Таблица 4.3 – Результаты замеров времени (данные не отсортированы)

Размер массива	Время, мкс		
	Поразрядная	Расческой	Шелла
1000	86.62	162.61	73.88
2000	182.43	369.90	174.06
3000	259.17	566.06	280.18
4000	338.33	780.25	374.35
5000	421.63	970.97	506.94
6000	500.53	1205.29	613.14
7000	580.47	1431.58	723.39
8000	678.11	1706.35	832.19
9000	756.42	1893.03	1019.44
10000	864.17	2215.13	1135.82
11000	959.53	2430.26	1262.90

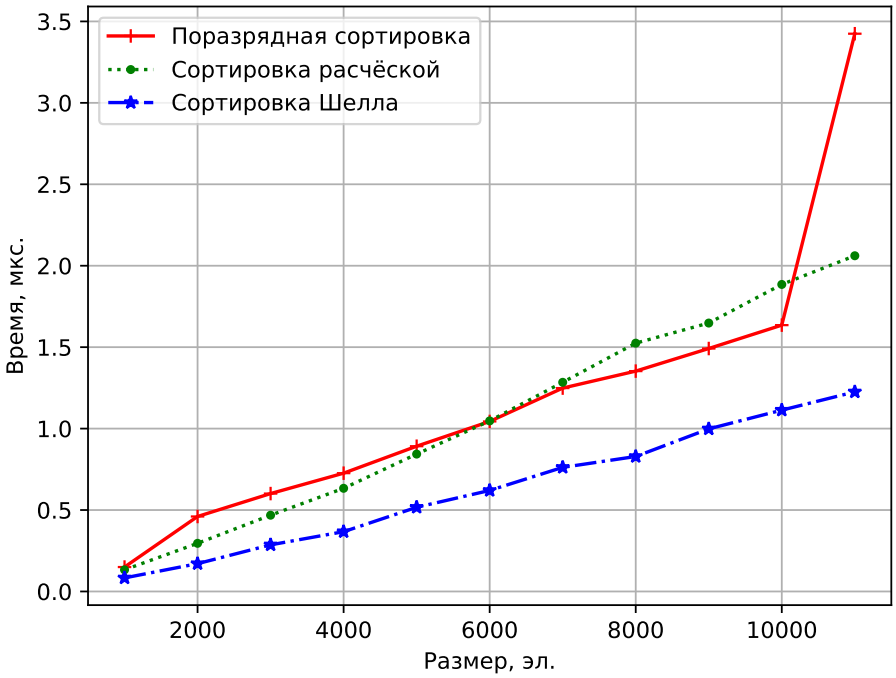

Рисунок 4.2 – Сравнение по времени алгоритмов сортировок на упорядоченном по возрастанию массиве

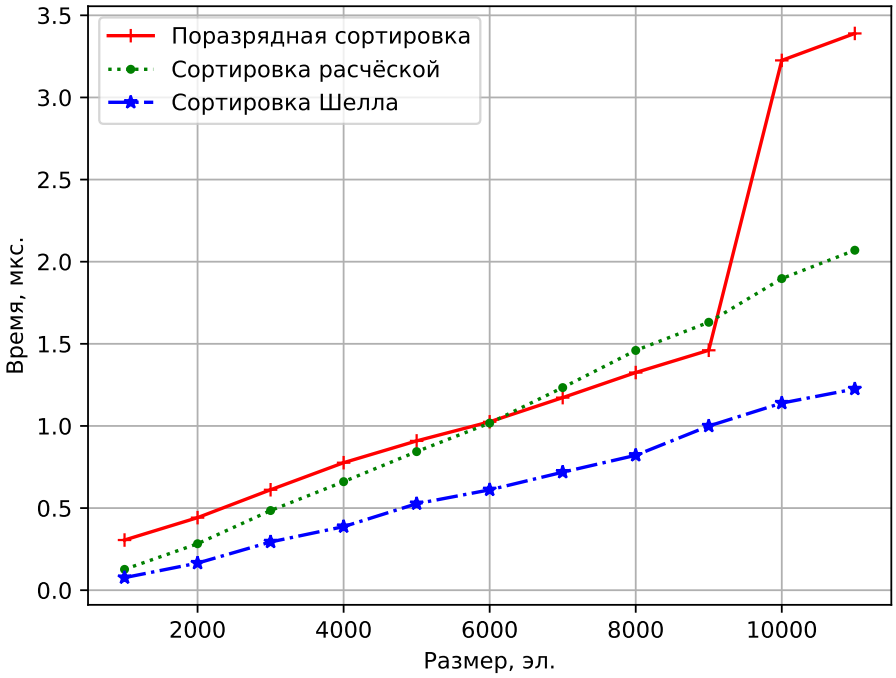

Рисунок 4.3 – Сравнение по времени алгоритмов сортировок на упорядоченном по убыванию массиве

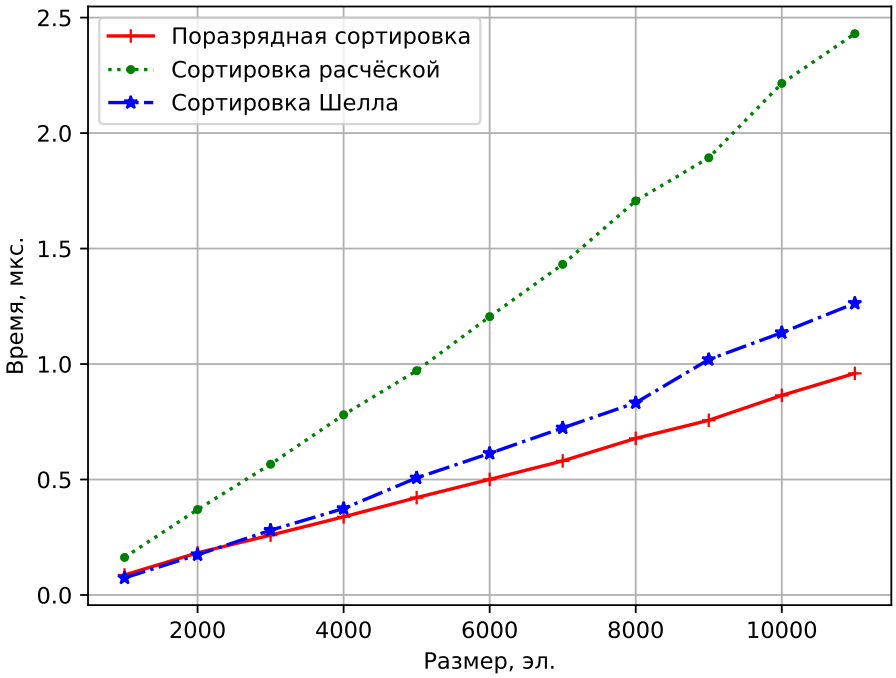

Рисунок 4.4 – Сравнение по времени алгоритмов сортировок на случайно заполненном массиве

4.4 Затраты по памяти реализаций алгоритмов

Затраты по памяти прдеставлены в таблице 4.4 и на рисунке 4.5.

Таблица 4.4 – Результаты замеров памяти

Размер массива	Память, байт		
	Поразрядная	Расческой	Шелла
1000	4056008.00	16.00	24.00
2000	16112008.00	16.00	24.00
3000	36168008.00	16.00	24.00
4000	64224008.00	16.00	24.00
5000	100280008.00	16.00	24.00
6000	144336008.00	16.00	24.00
7000	196392008.00	16.00	24.00
8000	256448008.00	16.00	24.00
9000	324504008.00	16.00	24.00
10000	400560008.00	16.00	24.00

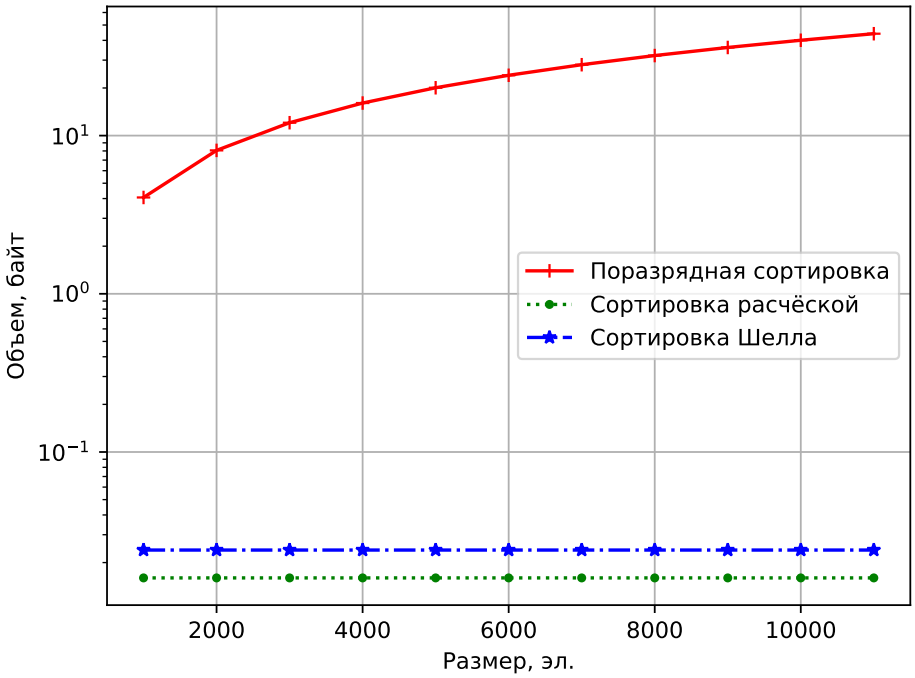

Рисунок 4.5 – Сравнение по времени алгоритмов сортировок на случайно заполненном массиве

Вывод

При сортировке упорядоченных массивов быстрее всех работает сортировка Шелла. Поразрядная сортировка неэффективна на упорядоченных данных из-за ограниченного количества уникальных разрядов, что снижает ее эффективность в этом случае, так как мало что изменится после первого прохода по разрядам, и большая часть времени будет затрачена на перестановку элементов. Сортировка расческой плохо работает на упорядоченных данных из-за специфики своего алгоритма, который проводит сортировку со слишком большими шагами, что может привести к ненужным перемещениям элементов и, следовательно, увеличению времени выполнения.

Ввиду того, что сортируемые числа неотрицательные и имеют малое число разрядов поразрядная сортировка оказалась самой эффективной по временным затратам при сортировке рандомных чисел.

Таким образом, на основе представленных данных можно сделать

вывод, что сортировка Шелла демонстрирует наилучшую производительность по сравнению с поразрядной сортировкой и сортировкой расческой.

Сортировки Шелла и расческой требуют меньше памяти, чем поразрядная, так как для последней необходимо создавать дополнительные массивы.

Заключение

Сортировка Шелла демонстрирует лучшую производительность при сортировке упорядоченных массивов. Поразрядная сортировка не эффективна из-за ограниченного числа уникальных разрядов, а сортировка расческой проводит сортировку слишком большими шагами, что увеличивает время выполнения из-за ненужных перемещений элементов.

Поразрядная сортировка оказалась эффективной только при сортировке неотрицательных чисел с небольшим числом разрядов.

Таким образом, сортировка Шелла продемонстрировала наилучшую производительность по сравнению с поразрядной сортировкой и сортировкой расческой, требуя при этом меньше памяти.

Цель данной лабораторной работы была достигнута, а именно были исследованы трудоемкости алгоритмов сортировки.

В результате выполнения лабораторной работы для достижения этой цели были выполнены следующие задачи:

1) описаны следующие алгоритмы сортировки:
   - поразрядная;
   - расческой;
   - Шелла;
2) релизованы описанные алгоритмы;
3) дана оценка трудоемкости алгоритмов;
4) дана оценка потребляемой памяти реализациями алгоритмов;
5) проведены замеры времени выполнения алгоритмов;

Список использованных источников

1 Д. Кнут. Искусство программирования для ЭВМ. Том 3. Сортировка ипоиск. — М.: ООО «И.Д. Вильямс», 2014. — С. 824.

2 Тема 3. Компьютерный анализ данных. Лекция 10. Методы и алгоритмы обработки и анализа данных [Электронный ресурс]. Режим доступа: `http://imamod.ru/~polyakov/arc/stud/mmca/lecture_10.pdf` (дата обращения: 04.12.2023).

3 Е.К. Липачев. Технология программирования. Методы сортировки данных. — Казань: Изд-во Казанского университета, 2017. — С. 19.

4 Документация по Microsoft C++ [Электронный ресурс]. — Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170&viewFallbackFrom=vs-2017` (дата обращения: 13.12.2023).