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 Иммореева М.А.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Строганов Д.В.

Москва — 2023 г.

Оглавление

Введение

Алгоритмы сортировки упорядочивают элементы в некоторой последовательности по определенному критерию. Существует множество алгоритмов сортировки, каждый из которых имеет свои преимущества и недостатки.

Выбор алгоритма сортировки зависит от множества факторов, таких как: размер данных, требования к производительности, доступная память. Анализ этих факторов помогает выбрать наиболее подходящий алгоритм сортировки для конкретной задачи.

1 Аналитическая часть

1.1 Цель и задачи

Целью данной работы является сравнение трех алгоритмов сортировки: плавная (smoothsort), сортировка перемешиванием и сортировка Шелла.

Для достижения поставленной цели необходимо решить следующие задачи:

— реализовать перечисленные алгоритмы сортировки;

— рассчитать трудоемкость реализованных алгоритмов;

— провести сравнительный анализ реализаций алгоритмов по затраченному процессорному времени, памяти.

1.2 Плавная сортировка

Плавная сортировка является модификацией алгоритма сортировки кучей. Данная сортировка сортирует массив путем создания кучи и многократного извлечения максимального элемента.

Основная идея плавной сортировки заключается в том, что кучи определяются с помощью чисел Леонардо, которые задаются следующим образом:

$$L(n) = \begin{cases} 1, \quad \text{если} \quad n = 0, \\ 1, \quad \text{если} \quad n = 1, \\ L(n-1) + L(n-2) + 1, \quad \text{если } \min n > 1. \end{cases} \tag{1.1}$$

K-ая куча Леонардо – это двоичное дерево с количеством вершин L(k), удовлетворяющее следующим условиям:

– число, записанное в корне, не меньше чисел в поддеревьях;

– левым поддеревом является (k - 1)-я куча Леонардо;

– правым – (k - 2)-я куча Леонардо.

Алгоритм плавной сортировки работает следующим образом.

1. В массиве записаны элементы, которые надо отсортировать.
2. Превращение массива в последовательность куч, кучи получены с помощью чисел Леонардо.
3. Повторяются шаги 2 и 3 до тех пор, пока расстояние не станет равным 0.
4. Пока последовательность куч не пустая, достаем максимальный элемент (это всегда корень самой правой кучи) и восстанавливаем порядок куч, который мог измениться.

1.3 Сортировка перемешиванием

Шейкерная сортировка, также известная как сортировка перемешиванием или сортировка коктейлем, является модификацией сортировки пузырьком.

Основная идея шейкерной сортировки заключается в том, чтобы проходить по массиву элементов в обоих направлениях, меняя местами соседние элементы, если они находятся в неправильном порядке. Это позволяет перемещать большие элементы в конец массива и малые элементы в начало массива, ускоряя процесс сортировки.

Алгоритм шейкерной сортировки работает следующим образом.

1. Устанавливается начальное значение флага, указывающего, были ли выполнены обмены на текущей итерации. Изначально флаг устанавливается в значение "ложь".

2. Выполняются следующие действия:

   – рассматриваются все элементы массива, начиная с первого и до последнего элемента;

   – сравниваются пары элементов, и если они находятся в неправильном порядке, они меняются местами, а флаг устанавливается в значение $True$;

   – рассматриваются все элементы массива в обратном порядке, начиная с последнего и до первого элемента;

   – сравниваются пары элементов, и если они находятся в неправильном порядке, они меняются местами, а флаг устанавливается в значение $True$.

3. Если флаг остается $False$ после выполнения шага 2, значит массив уже отсортирован и сортировка завершается. Иначе повтор шага 2.

1.4 Сортировка Шелла

Сортировка Шелла является алгоритмом сортировки, который представляет собой модификацию сортировки вставками.

Основная идея сортировки Шелла заключается в том, чтобы предварительно сортировать элементы массива, находящиеся на определенном расстоянии друг от друга, а затем постепенно уменьшать это расстояние и повторять процесс сортировки. Это позволяет перемещать элементы на более короткие расстояния и ускоряет процесс сортировки.

Каждый проход в алгоритме характеризуется смещением h_i, таким, что сортируются элементы отстающие друг от друга на h_i позиций. Шелл предлагал использовать $h_t = N/2$, $h_{t-1} = h_t/2$, ..., $h_0 = 1$. Возможны и другие смещения, но $h_0 = 1$ всегда.

Алгоритм плавной сортировки работает следующим образом:

1. Шаг 0. $i = t$.
2. Шаг 1. Разобьем массив на списки элементов, отстающих друг от друга на h_i. Таких списков будет h_i.
3. Шаг 2. Отсортируем элементы каждого списка сортировкой вставками.
4. Шаг 3. Объединим списки обратно в массив. Уменьшим i. Если i неотрицательно – вернемся к шагу 1.

Вывод

В данном разделе были рассмотрены алгоритмы сортировки: плавная (smoothsort), сортировка перемешиванием и сортировка Шелла.

2 Конструкторская часть

В этом разделе приведены схемы алгоритмов сортировки: плавная (smoothsort), сортировка перемешиванием и сортировка Шелла. Проведен анализ трудоемкости данных алгоритмов. Приведены требования к программному обеспечению.

2.1 Разработка алгоритмов

На рисунке 2.1 приведена схема алгоритма плавной сортировки (smoothsort). На рисунке 2.2 приведена схема алгоритма сортировки перемешиванием. На рисунке 2.3 приведена схема алгоритма сортировки Шелла.

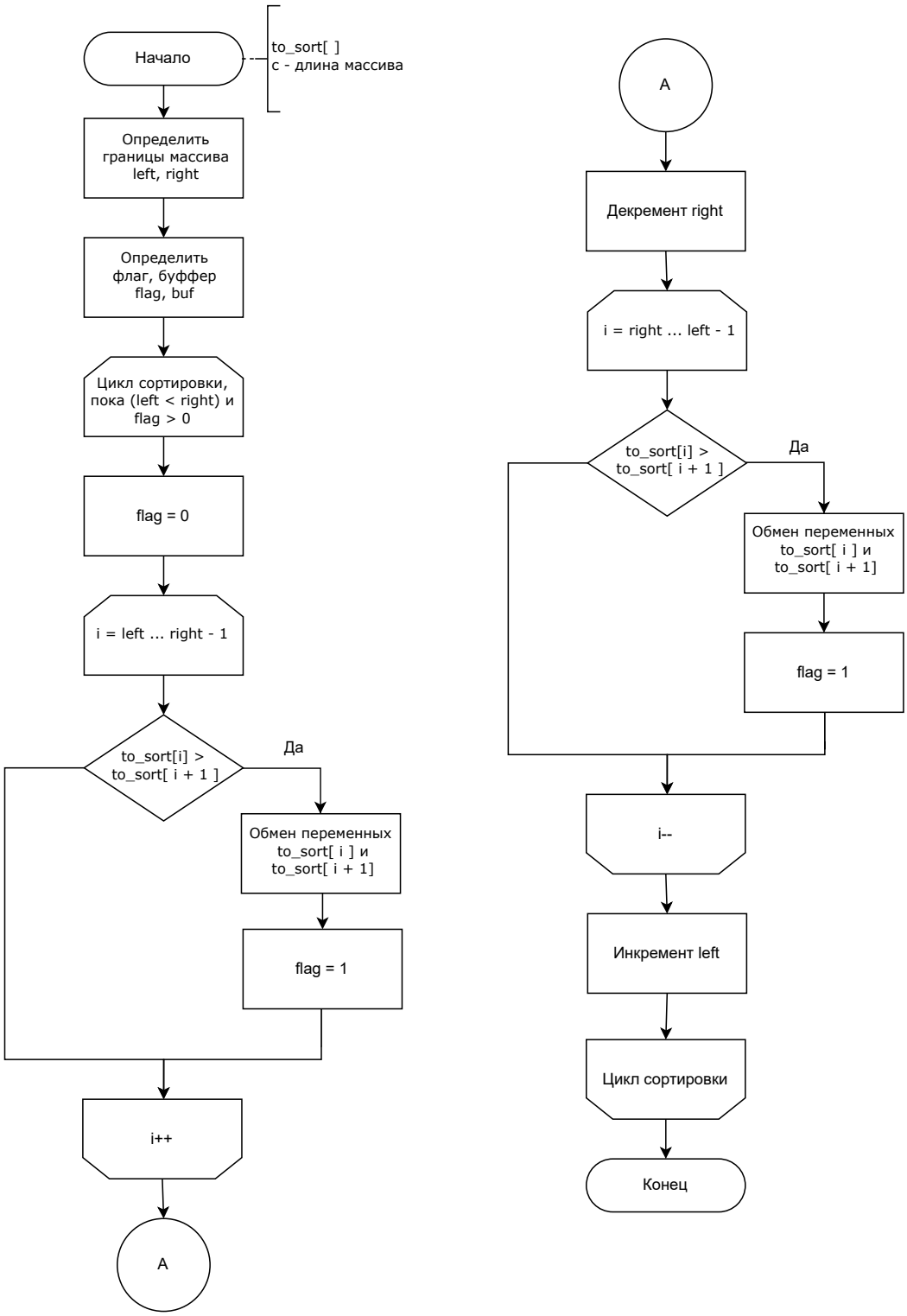

Рисунок 2.1 – Схема алгоритма сортировки перемешиванием

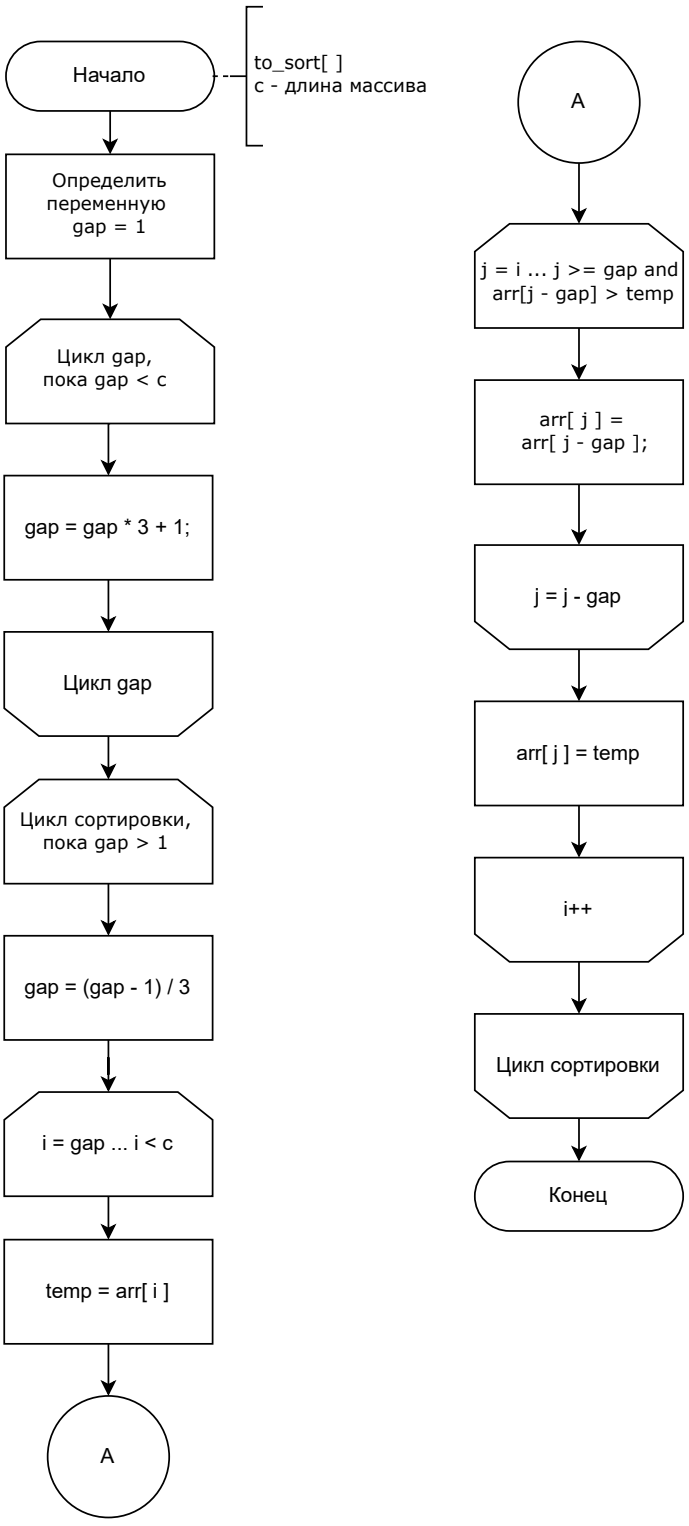

Рисунок 2.2 – Схема алгоритма плавной сортировки (smoothsort)

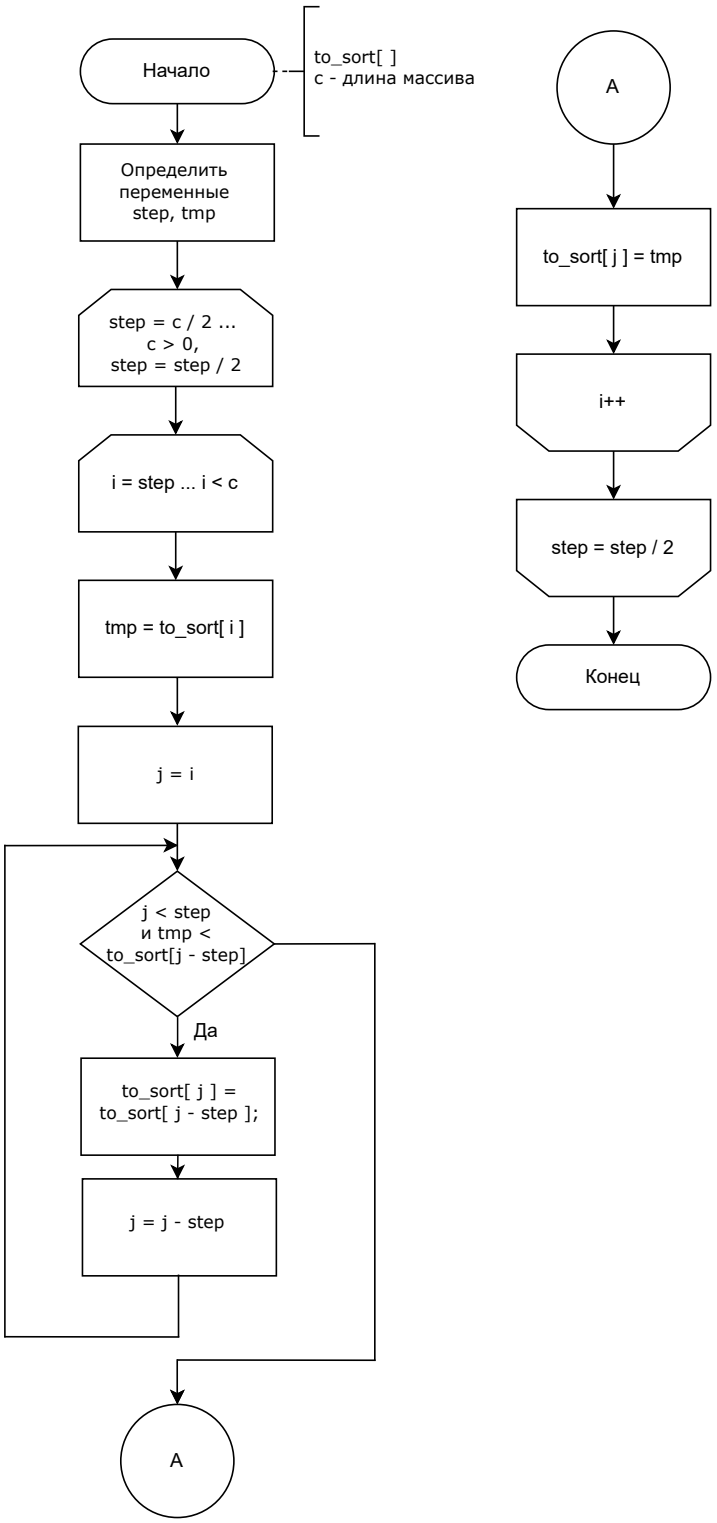

Рисунок 2.3 – Схема алгоритма сортировки Шелла

2.2 Требования к вводу

К вводу программы прилагаются следующие требования.

1. Перед вводом массива запрашиваются его размерность.
2. На вход подается строка из чисел.
3. Считываются числа типа int.

2.3 Требования к программе

К программе предъявляется требования: на выходе программы — массив с результатом сортировки исходного массива.

2.4 Трудоемкость алгоритмов

Трудоемкость вычисляется на основании следующей модели вычислений:

1. бзовые операции, имеющие стоимость 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [ ], получение полей класса;
2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.1);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.1}$$

3. трудоемкость цикла рассчитывается, как (2.2);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.2}$$

4. стоимость условного перехода равна 0, стоимость вычисления условия остаётся.

2.4.1 Плавная сортировка

Данные о трудоемкости плавной сортировки были взяты из источника [3].

Лучший случай: массив отсортирован; сортировка кучи во вложенном цикле не происходит.

Трудоемкость: $O(n)$

Худший случай: массив отсортирован в обратном порядке; вложенный цикл отсортирует всю кучу.

Трудоемкость: $O(n \log n)$

2.4.2 Сортировка перемешиванием

Лучший случай: массив отсортирован; не произошло ни одного обмена за 1 проход -> выход из цикла

Худший случай: массив отсортирован в обратном порядке; в каждом случае происходил обмен, все внутренние циклы будут состоять из |left - right| итераций, где left, right – индексы переменных, которые являются границей сортируемой части массива в очередном цикле.

Трудоёмкость сравнения внешнего цикла $while(swap == True)$ равна (2.3):

$$f_{outer} = 1 + 2 \cdot (N - 1) \tag{2.3}$$

Суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N-1]$, равна (2.4):

$$f_{inner} = 5(N - 1) + \frac{2 \cdot (N - 1)}{2} \cdot (3 + f_{if}) \tag{2.4}$$

Трудоёмкость условия во внутреннем цикле равна (2.5):

$$f_{if} = 4 + \begin{cases} 0, & \text{лучший случай,} \\ 9, & \text{худший случай.} \end{cases} \tag{2.5}$$

Трудоёмкость в лучшем случае (2.6):

$$f_{best} = -3 + \frac{3}{2}N \approx \frac{3}{2}N = O(N) \tag{2.6}$$

Трудоёмкость в худшем случае (2.7):

$$f_{worst} = -3 - 8N + 8N^2 \approx 8N^2 = O(N^2) \tag{2.7}$$

2.4.3 Сортировка Шелла

Лучший случай: Массив отсортирован; обмена не происходит во внутреннем цикле и он завершается на первой итерации при каждом вызове;

Худший случай: Массив отсортирован в обратном порядке; Неверно выбраны промежутки; Во внутреннем цикле каждый раз происходит сдвиг по шагу;

Трудоемкость в лучшем случае выведена формуле (2.8).

$$\begin{gathered} f_{best} = 3 + 4 + \frac{N}{4} \cdot (3 + 2 + log(N) \cdot (2 + 4 + 4)) = \\ = 7 + \frac{5N}{4} + \frac{5N \cdot log(N)}{2} = O(N \cdot log(N)) \end{gathered} \tag{2.8}$$

Трудоемкость в худшем случае выведена формуле (2.9).

$$\begin{gathered} f_{worst} = 7 + \frac{N}{4} \cdot (5 + log(N) \cdot (10 + log(N) \cdot (6 + 4))) = \\ = 7 + \frac{5N}{4} + \frac{5N \cdot log(N)}{2} + \frac{5N \cdot log^2(N)}{2} = O(N \cdot log^2(N)) \end{gathered} \tag{2.9}$$

Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов, были приведены требования к программному обеспечению, была вычислена трудоемкость алгоритмов. Таким образом, были получены следующие данные о трудоемкости:

- Плавная сортировка: лучший – $O(n)$, худший – $O(n \log n)$;
- Сортировка перемешиванием: лучший – $O(n)$, худший – $O(n^2)$;
- Сортировка Шелла: лучший – $O(n \log n)$, худший – $O(n \log^2 n)$.

3 Технологическая часть

В данном разделе будут приведены средства реализации программы и листинги кода.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования C. Выбор обусловлен наличием библиотек для измерения времени, наличием инструментов для работы с массивами.

3.2 Сведения о модулях программы

Программа состоит из следующих модулей: main.c - главный файл программы, в котором располагается вся программа, time.c - файл программы с замерами времени.

3.3 Реализация алгоритмов

В листингах 3.1, 3.2, 3.3, приведены реализации алгоритмов сортировки массивов.

Листинг 3.1 – Плавная сортировка.

```
1     void SmoothSort(int *arr, int c) {
2         int gap = 1;
3         while (gap < c) {
4             gap = gap * 3 + 1;
5         }
6
7         while (gap > 1) {
8             gap = (gap - 1) / 3;
9
10            for (int i = gap; i < c; i++) {
11                int temp = arr[i];
12                int j;
13
14                for (j = i; j >= gap && arr[j - gap] > temp; j -=
                      gap) {
15                    arr[j] = arr[j - gap];
16                }
17
18                arr[j] = temp;
19            }
20        }
21    }
```

Листинг 3.2 – Сортировка перемешиванием

```
1     void ShakerSort(int *to_sort, int c)
2     {
3         int left = 0, right = c - 1;
4         int flag = 1, t;
5         while ((left < right) && flag > 0) {
6             flag = 0;
7             for (int i = left; i < right; i++) {
8                 if (to_sort[i] > to_sort[i + 1])
9                 {
10                    t = to_sort[i];
11                    to_sort[i] = to_sort[i + 1];
12                    to_sort[i + 1] = t;
13                    flag = 1;
14                }
15            }
16            right--;
17            for (int i = right; i > left; i--) {
18                if (to_sort[i - 1] > to_sort[i])
19                {
20                    int t = to_sort[i];
21                    to_sort[i] = to_sort[i - 1];
22                    to_sort[i - 1] = t;
23                    flag = 1;
24                }
25            }
26            left++;
27        }
28    }
```

Листинг 3.3 – Сортировка Шелла

```
1     void ShellSort(int *to_sort, int c)
2     {
3         int i, j, step;
4         int tmp;
5         for (step = c / 2; step > 0; step /= 2)
6         for (i = step; i < c; i++) {
7             tmp = to_sort[i];
8             for (j = i; j >= step; j -= step)
9             {
10                if (tmp < to_sort[j - step])
11                to_sort[j] = to_sort[j - step];
12                else
13                break;
14            }
15            to_sort[j] = tmp;
16        }
17    }
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов сортировки массивов.

Таблица 3.1 – Тестирование функций

Массив	Ожидаемый результат
$(1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9)$	$(1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9)$
$(9\ \ 8\ \ 7\ \ 6\ \ 5\ \ 4\ \ 3\ \ 2\ \ 1)$	$(1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9)$
$(1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1)$	$(1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1)$
$(1\ \ 1\ \ 2\ \ 2\ \ 8\ \ 8\ \ 3\ \ 3\ \ -1\ \ 4)$	$(-1\ \ 1\ \ 1\ \ 2\ \ 2\ \ 3\ \ 3\ \ 4\ \ 8\ \ 8)$
$(1\ \ 1\ \ 2\ \ 2\ \ 8\ \ 8\ \ 3\ \ 3\ \ -1\ \ -2)$	$(-2\ \ -1\ \ 1\ \ 1\ \ 2\ \ 2\ \ 3\ \ 3\ \ 8\ \ 8)$
$(-1\ \ -3\ \ -5\ \ -2)$	$(-5\ \ -3\ \ -2\ \ -1)$
(-1)	(-1)

Тестирование пройдено успешно всеми реализациями алгоритмов.

Вывод

Были разработаны и протестированы алгоритмы: плавная сортировка, сортировка перемешиванием, сортировка Шелла.

4 Исследовательская часть

В данном разделе будет приведена постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся эксперимент по замеру времени:

— 64-разрядная операционная система, процессор x64;

— память 16 Гб;

— процессор Intel(R) Core(TM) i7–4700HQ CPU @ 2.40 ГГц.

Во время выполнения эксперимента ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно программой.

4.2 Расчет затрат памяти

Пусть с – размер сортируемого массива, экспериментгогда затраты памяти на приведенные выше алгоритмы будут следующими:

Сортировка перемешиванием:

– Массив – с * sizeof(int);

– Переменные – 5 * sizeof(int).

Сортировка Шелла:

– Массив – с * sizeof(int);

– Переменные – 5 * sizeof(int).

Плавная сортировка:

– Массив – с * sizeof(int);

– Переменные – 3 * sizeof(int).

В итоге, сортировка перемешиванием и сортировка Шелла занимают одинаковое количество памяти. Плавная сортировка выигрывает по памяти всего на размер двух переменных типа int.

4.3 Время выполнения алгоритмов

Время выполнения алгоритмов было замерено при помощи функции clock() из библиотеки <time.h> языка C. Данная функция возвращает количество временных тактов с момента запуска программы, вызвавшей функцию clock().

Замеры времени для каждой реализации сортировки проводились 100 раз. В качестве результата взято среднее время работы реализации алгоритма.

Результаты замеров приведены в таблицах 4.1, 4.2, 4.3 (время в мс).

Таблица 4.1 – Результаты замеров времени на случайных данных(мс.)

Размерность массива	Сортировка перемешиванием	Сортировка Шелла	Плавная сортировка
500	0.000 480	0.000 020	0.000 010
1000	0.001 510	0.000 110	0.000 090
1500	0.003 550	0.000 070	0.000 120
2000	0.005 940	0.000 120	0.000 170
2500	0.009 570	0.000 220	0.000 200
3000	0.014 710	0.000 230	0.000 140
3500	0.019 260	0.000 300	0.000 210
4000	0.026 680	0.000 330	0.000 370
4500	0.033 320	0.000 470	0.000 300
5000	0.040 930	0.000 470	0.000 390

На изображении 4.1 приведен график на основе табличных данных из 4.1.

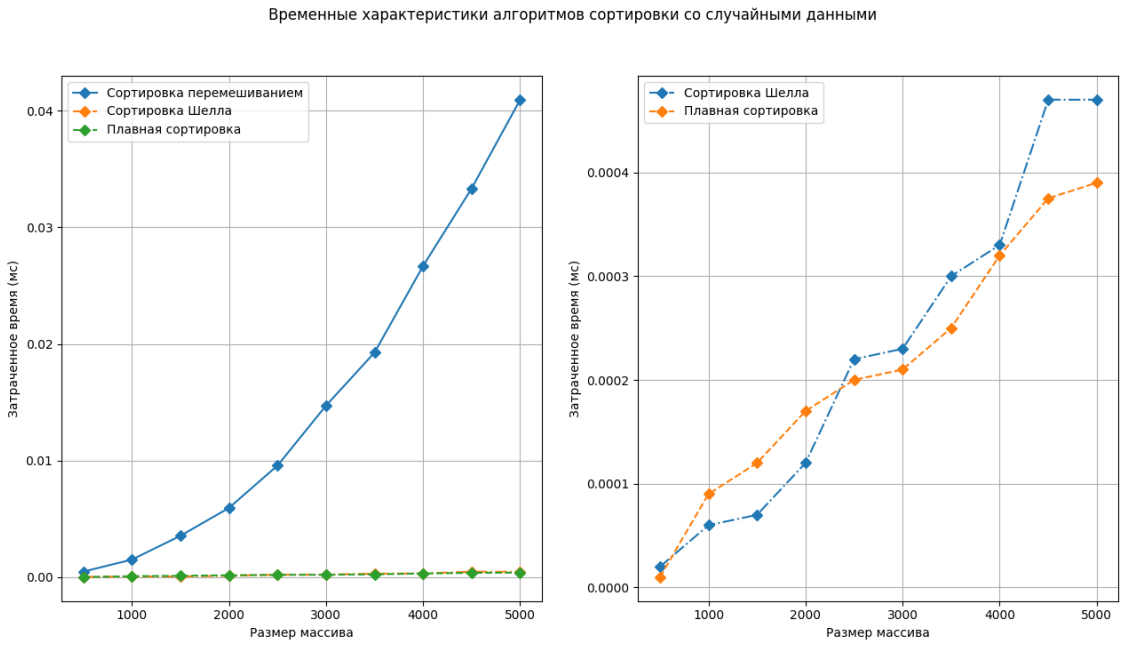

Рисунок 4.1 – График результата замеров времени на случайных данных

Как видно из данных с случайным заполнением массива, сортировка перемешиванием заняла больше времени чем остальные сортировки с таким объемом неотсортированных данных; плавная сортировка и сортировка Шелла значительно эффективнее по затраченному времени работы сортировки перемешиванием, плавная сортировка быстрее остальных.

Таблица 4.2 – Результаты замеров времени на отсортированных данных(мс.)

Размерность массива	Сортировка перемешиванием	Сортировка Шелла	Плавная сортировка
500	0.000 020	0.000 000	0.000 010
1000	0.000 030	0.000 050	0.000 010
1500	0.000 040	0.000 070	0.000 030
2000	0.000 080	0.000 070	0.000 060
2500	0.000 110	0.000 140	0.000 070
3000	0.000 160	0.000 160	0.000 070
3500	0.000 210	0.000 190	0.000 070
4000	0.000 290	0.000 180	0.000 120
4500	0.000 360	0.000 220	0.000 120
5000	0.000 460	0.000 240	0.000 150

На изображении 4.2 приведен график на основе табличных данных из 4.2.

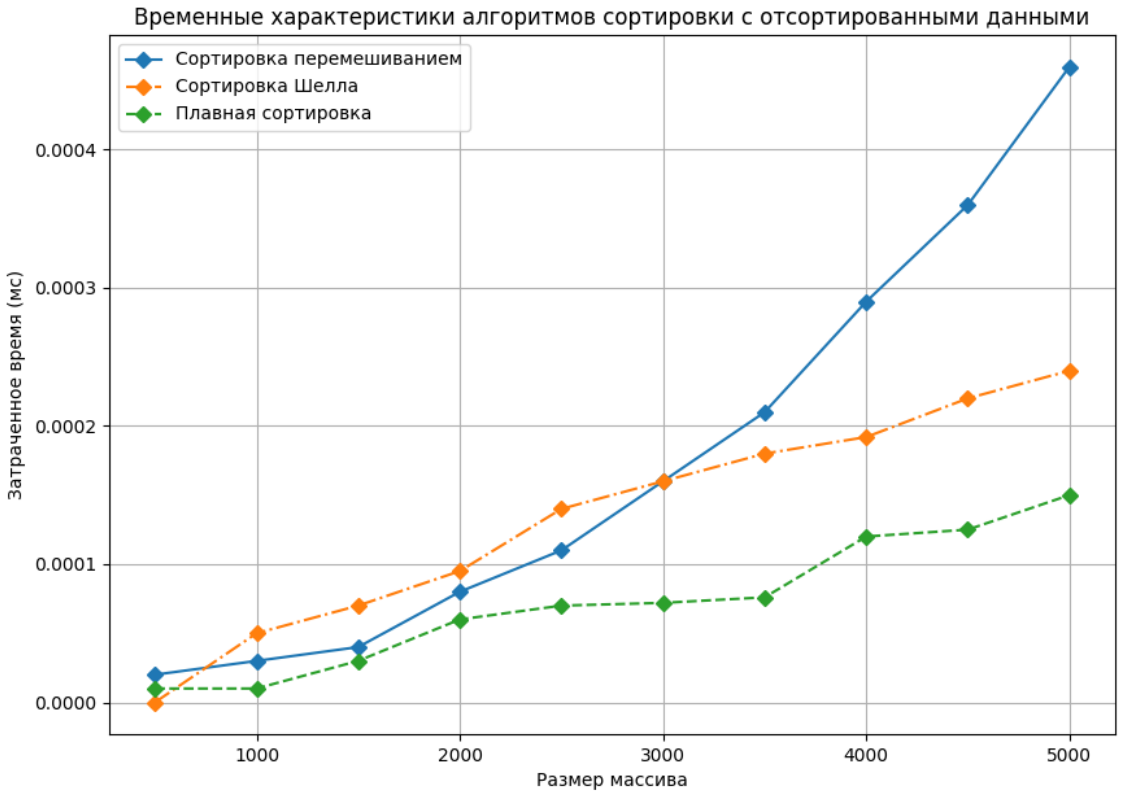

Рисунок 4.2 – График результата замеров времени на отсортированных данных

Как видно из данных с отсортированным заполнением массива, плавная сортировка быстрее всех сортировок справлялась с данными.

Таблица 4.3 – Результаты замеров времени на отсортированных в обратном порядке данных(мс.)

Размерность массива	Сортировка перемешиванием	Сортировка Шелла	Плавная сортировка
500	0.000 390	0.000 020	0.000 010
1000	0.001 540	0.000 090	0.000 030
1500	0.003 280	0.000 120	0.000 040
2000	0.005 990	0.000 120	0.000 050
2500	0.009 460	0.000 140	0.000 050
3000	0.013 000	0.000 160	0.000 060
3500	0.017 650	0.000 200	0.000 090
4000	0.023 050	0.000 200	0.000 150
4500	0.028 870	0.000 240	0.000 220
5000	0.036 060	0.000 250	0.000 220

На изображении 4.3 приведен график на основе табличных данных из 4.3. Как видно из данных с отсортированным в обратном порядке запол-

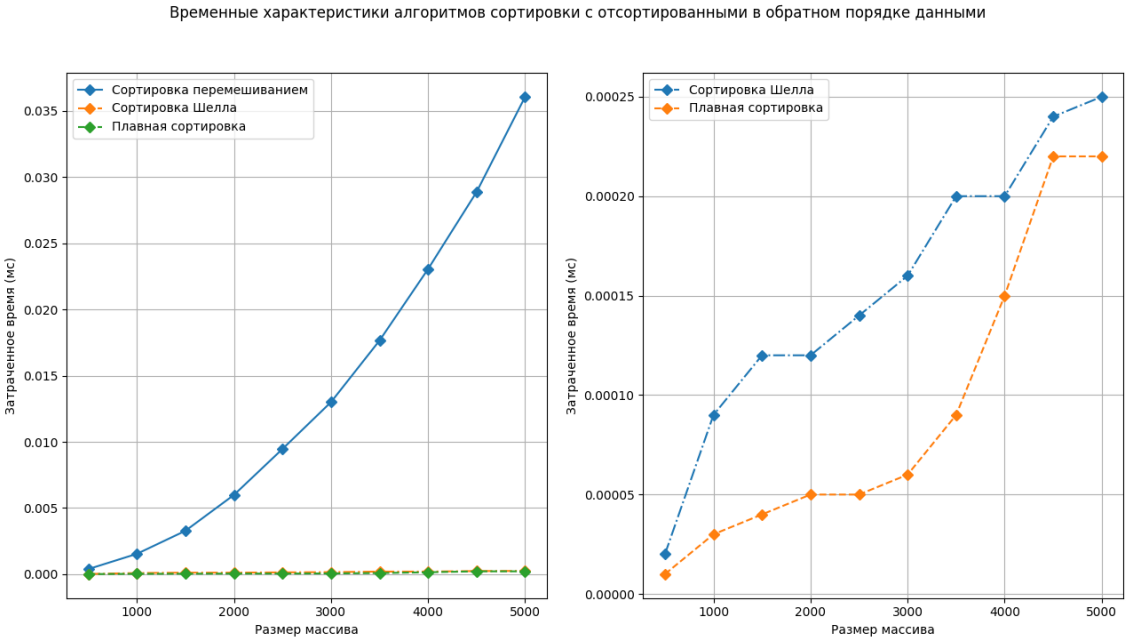

Рисунок 4.3 – График результата замеров времени на отсортированных в обратном порядке данных

нением массива, сортировка перемешиванием справляется медленнее всех из рассматриваемых алгоритмов с данным объемом данных. Сортировка Шелла и Плавная сортировка в 150 раз быстрее сортировки перемешиванием на замерах от 500 вхождений в сортируемом массиве до 5000, плавная сортировка выигрывает по времени у сортировки Шелла.

Вывод

Из проведенного анализа сделаны следующие выводы: все три сортировки демонстрируют высокую производительность при работе с уже отсортированными данными, в каждой сортировке осуществляется только один проход по циклам; сортировка перемешиванием заняла больше всего времени при отсортированных в обратном порядке данных $O(n^2)$, в отличии от плавной сортировки $O(N \log N)$, которая была самой быстрой среди рассматриваемых; во время анализа замеров времени при случайных данных сортировка перемешиванием также заняла больше всего времени, плавная сортировка – лидер по эффективности выполнения.

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

— были реализованы алгоритмы сортировки Шелла, премешиванием, плавная;

— был произведён анализ трудоёмкости алгоритмов;

— был сделан сравнительный анализ алгоритмов на основе экспериментальных данных по времени, памяти.

Выбор алгоритма сортировки зависит от факторов, являющихся критичными в поставленной задаче. Анализ этих факторов помогает выбрать наиболее подходящий алгоритм сортировки для конкретной задачи.

Из исследовательской части сделаны следующие выводы: все три сортировки демонстрируют высокую производительность при работе с уже отсортированными данными, в каждой сортировке осуществляется только один проход по циклам. Сортировка перемешиванием заняла больше всего времени при отсортированных в обратном порядке данных $O(n^2)$, в отличии от плавной сортировки $O(N \log N)$, которая была самой быстрой среди рассматриваемых. Во время анализа замеров времени при случайных данных сортировка перемешиванием также заняла больше всего времени, плавная сортировка – лидер по быстроте выполнения. Сортировка перемешиванием и сортировка Шелла занимают одинаковое количество памяти. Плавная сортировка выигрывает по памяти всего на размер двух переменных типа int.

Список использованных источников

1. Кормен Т. Алгоритмы: построение и анализ [Текст] / Кормен Т. - Вильямс, 2014. - 198 с. - 219 с.

2. Основные виды сортировок и примеры их реализации // Академия Яндекса URL: https://academy.yandex.ru/journal/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii (дата обращения: 16.10.2023).

3. Smoothsort Demystified URL: https://www.keithschwarz.com/smoothsort/(дата обращения 27.10.2023) .