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: «Трудоёмкость сортировок»

Студент ИУ7-51Б (Группа) _____ (Подпись, дата) А. А. Кузин (И. О. Фамилия)

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

2023 г.

Содержание

Введение

Сортировка — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент в списке имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

Целью данной лабораторной работы является анализ алгоритмов сортировок.

Необходимо выполнить следующие **задачи**:

1. Ознакомиться со следующими алгоритмами сортировок:
   - Плавная сортировка (smoothsort);
   - Сортировка перемешиванием;
   - Сортировка Шелла.
2. Реализовать алгоритмы сортировок;
3. Выполнить замеры затрат реализаций алгоритмов по памяти;
4. Выполнить замеры затрат реализаций алгоритмов по процессорному времени;
5. Провести сравнительных анализ алгоритмов.

1 Аналитическая часть

1.1 Плавная сортировка (smoothsort)

Плавная сортировка (англ. Smooth sort) — модификация сортировки кучей, разработанный Э. Дейкстрой. Как и пирамидальная сортировка, в худшем случае работает за время $\Theta(N \log N)$. Преимущество плавной сортировки в том, что её время работы приближается к $O(N)$, если входные данные частично отсортированы, в то время как у сортировки кучей время работы не зависит от состояния входных данных.

1.2 Сортировка перемешиванием

Сортировка перемешиванием [1] – это разновидность сортировки пузырьком. Отличие в том, что данная сортировка в рамках одной итерации проходит по массиву в обоих направлениях (слева направо и справа налево), тогда как сортировка пузырьком – только в одном направлении (слева направо).

Общие идеи алгоритма:

- обход массива слева направо, аналогично пузырьковой – сравнение соседних элементов, меняя их местами, если левое значение больше правого;
- обход массива в обратном направлении (справа налево), начиная с элемента, который находится перед последним отсортированным, то есть на этом этапе элементы также сравниваются между собой и меняются местами, чтобы наименьшее значение всегда было слева.

1.3 Сортировка Шелла

Метод предложен в 1959 году и назван по имени автора метода Дональда Шелла (Donald Shell).

Сортировка Шелла [2—4] (англ. Shell Sort) является улучшением сортировки вставками. Часто еще называемый "сортировка вставками с уменьшением расстояния". Основная идея этого метода заключается в том, чтобы в начале устранить массовый беспорядок в массиве, сравнивая далеко отстоящие друг от друга элементы. Постепенно интервал между сравниваемыми

элементами уменьшается до единицы. Это означает, что на поздних стадиях сортировка сводится просто к перестановкам соседних элементов (если, конечно, такие перестановки являются необходимыми).

Пусть d - интервал между сравниваемыми элементами. Первоначально используемая Шеллом последовательность длин промежутков: $d_1 = \frac{N}{2}, d_i = \frac{d_{i-1}}{2}, \ldots d_k = 1$. Процесс завершается обычной сортировкой вставками получившегося списка.

Вывод

В данном разделе были рассмотрены следующие алгоритмы сортировок:

- Плавная сортировка;
- Сортировка перемешиванием;
- Сортировка Шелла.

2 Конструкторская часть

В данном разделе будут приведены псевдокоды алгоритмов сортировок, описание используемых типов данных и структуры программного обеспечения.

К программе предъявлен ряд функциональных требований:

- наличие интерфейса для выбора действий;
- возможность ввода чисел, используя ввод с клавиатуры;
- возможность произвести замеры процессорного времени работы реализованных алгоритмов сортировок.

2.1 Разработка алгоритмов

В листинге 2.1 представлен псевдокод сортировки перемешиванием.

В листинге 2.2 представлен псевдокод сортировки Шелла.

В листинге 2.3 представлен псевдокод плавной сортировки.

Листинг 2.1 – псевдокод сортировки перемешиванием

```
1  procedure sort_shaker(var arr: array of Integer);
2  begin
3    m := Length(arr)
4    i := 0
5    while i < m do
6    begin
7      j := i + 1
8      while j < m do
9      begin
10       if arr[j] < arr[j-1] then
11       begin
12         Swap(arr[j], arr[j-1])
13       end
14       j := j + 1
15     end
16     m := m - 1
17     k := m - 1
18     while k > i do
19     begin
20       if arr[k] < arr[k-1] then
21       begin
```

```
22          Swap(arr[k], arr[k-1])
23        end
24        k := k - 1
25      end
26      i := i + 1
27    end
28  end
```

Листинг 2.2 – псевдокод сортировки Шелла

```
1  procedure sort_shell(var arr: array of Integer)
2  begin
3    n := Length(arr)
4    gap := n div 2
5    while gap > 0 do
6    begin
7      for i := gap to n - 1 do
8      begin
9        temp := arr[i]
10       j := i
11       while (j >= gap) and (arr[j - gap] > temp) do
12       begin
13         arr[j] := arr[j - gap]
14         j := j - gap
15       end
16       arr[j] := temp
17     end
18     gap := gap div 2
19   end
20 end
```

Листинг 2.3 – псевдокод плавной сортировки

```
1  function smooth_sort(arr: array of Integer): array of Integer
2    function leonardo(k: Integer): Integer
3    begin
4      if k < 2 then
5        Result := 1
6      else
7        Result := leonardo(k - 1) + leonardo(k - 2) + 1
8    end
9
10   procedure heapify(start, ending: Integer);
11   begin
```

```
12      i := start
13      j := 0
14      k := 0
15      while k < ending - start + 1 do
16      begin
17        if k and $AAAAAAAA <> 0 then
18        begin
19          j := j + i
20          i := i shr 1
21        end
22        else
23        begin
24          i := i + j
25          j := j shr 1
26        end
27        k := k + 1
28      end
29      while i > 0 do
30      begin
31        j := j shr 1
32        k := i + j
33        while k < ending do
34        begin
35          if arr[k] > arr[k - i] then
36            break
37          arr[k] := arr[k - i]
38          arr[k - i] := arr[k]
39          k := k + i
40        end;
41        i := j
42      end
43    end
44
45  begin
46    n := Length(arr)
47    p := n - 1
48    q := p
49    r := 0
50    while p > 0 do
51    begin
52      if (r and $03) = 0 then
```

```
53        heapify(r, q)
54      if leonardo(r) = p then
55        r := r + 1
56      else
57      begin
58        r := r - 1
59        q := q - leonardo(r)
60        heapify(r, q)
61        q := r - 1
62        r := r + 1
63      end
64      arr[0] := arr[p]
65      arr[p] := arr[0]
66      p := p - 1
67    end
68    for i := 0 to n - 2 do
69    begin
70      j := i + 1
71      while (j > 0) and (arr[j] < arr[j - 1]) do
72      begin
73        arr[j] := arr[j - 1]
74        arr[j - 1] := arr[j]
75        j := j - 1
76      end
77    end
78    Result := arr
79  end
```

2.2 Описание используемых типов данных

При реализации алгоритмов будет использован *массив* — упорядоченный набор элементов `целочисленного типа`.

2.3 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.

2.4 Трудоемкость алгоритмов

В следующих частях будут приведены рассчеты трудоемкостей алгоритмов сортировок.

2.4.1 Плавная сортировка (smoothsort)

На первом этапе перебирается N элементов, добавляя его в уже имеющиеся слева кучи. Добавление в кучу обходится в

$$f_{elem} = O(1) \tag{2.5}$$

затем для кучи нужно сделать просейку.

В неупорядоченных данных просейка для каждого добавления обходится

в:

$$f_{unsort} = O(log(N)) \tag{2.6}$$

так как из-за случайных чисел просейке приходится проходить уровни дерева часто до самого низа.

Поэтому, на первом этапе наилучшая сложность по времени: для упорядоченных чисел:

$$f_{sort} = O(N) \tag{2.7}$$

для случайных чисел:

$$f_{unsort} = O(N \cdot log(N)) \tag{2.8}$$

Для второго этапа ситуация аналогичная. При обмене очередного максимума опять необходимо просеять кучу, в корне которой он находился.

На втором этапе наилучшая сложность по времени такая же как и на первом: для упорядоченных чисел:

$$f_{sort} = O(N) \tag{2.9}$$

для случайных чисел:

$$f_{unsort} = O(N \cdot log(N)) \tag{2.10}$$

Складывая временные сложности для первого и второго этапа: для упорядоченных чисел:

$$f_{best} = O(2 \cdot N) = O(N) \tag{2.11}$$

В общем, худшая и средняя временная сложность для плавной сортировки:

$$f_{worst} = O(2 \cdot N \cdot log(N)) = O(N \cdot log(N)) \tag{2.12}$$

2.4.2 Сортировка перемешиванием

- Трудоёмкость сравнения внешнего цикла $while$, которая вычисляется по формуле (2.13).

$$f_{outer} = 1 + 2 \cdot (N - 1). \tag{2.13}$$

- Суммарная трудоёмкость внутренних циклов, количество итераций кото-

рых меняется в промежутке $[1..N-1]$, которая вычисляется по формуле (2.14).

$$f_{inner} = 5(N-1) + \frac{2 \cdot (N-1)}{2} \cdot (3 + f_{if}). \tag{2.14}$$

- Трудоёмкость условия во внутреннем цикле, которая вычисляется по формуле (2.15).

$$f_{if} = 4 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases} \tag{2.15}$$

Трудоёмкость в **лучшем** случае (2.16):

$$f_{best} = -3 + \frac{3}{2}N + \approx \frac{3}{2}N = O(N). \tag{2.16}$$

Трудоёмкость в **худшем** случае (2.17):

$$f_{worst} = -3 - 8N + 8N^2 \approx 8N^2 = O(N^2). \tag{2.17}$$

2.4.3 Сортировка Шелла

Трудоемкость в лучшем случае при отсортированном массиве, когда ничего не обменивается, но все же данные рассматриваются. Выведена в формуле (2.18).

$$\begin{gathered} f_{best} = 3 + 4 + \frac{N}{4} \cdot (3 + 2 + log(N) \cdot (2 + 4 + 4)) = \\ = 7 + \frac{5N}{4} + \frac{5N \cdot log(N)}{2} = O(N \cdot log(N)) \end{gathered} \tag{2.18}$$

Трудоемкость в худшем случае при отсортированном массиве в обратном порядке. Выведена в формуле (2.19).

$$\begin{gathered} f_{worst} = 7 + \frac{N}{4} \cdot (5 + log(N) \cdot (10 + log(N) \cdot (6 + 4))) = \\ = 7 + \frac{5N}{4} + \frac{5N \cdot log(N)}{2} + \frac{5N \cdot log^2(N)}{2} = O(N \cdot log^2(N)) \end{gathered} \tag{2.19}$$

Вывод

В данном разделе на основе теоретических данных были перечислены требования к ПО, сделаны псевдокоды реализуемых алгоритмов и представлена модель вычислений на основе данных, полученных на этапе анализа.

При отсортированном массиве асимптотическая сложность алгоритмов плавной сортировки и сортировки перемешиванием стремится к $O(N)$, в то время как сортировка шелла к $O(N)$.

Самыми эффективными по трудоемкости при неотсортированном массиве являются сортировки перемешиваем и Шелла, имеющие одинаковую асимптотическую сложность - $O(N \cdot log(N))$

3 Технологическая часть

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

3.1 Средства реализации

В качестве языка программирования, используемого при написании данной лабораторной работы, был выбран C++ [5], так как в нем имеется контейнер `std::string`, представляющий собой массив символов `char`, и библиотека `<ctime>` [6], позволяющая производить замеры процессорного времени.

В качестве среды для написания кода был выбран *Visual Studio Code* за счет того, что она предоставляет функционал для проектирования, разработки и отладки ПО.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

- `main.cpp` — точка входа программы, пользовательское меню;
- `general` — модуль с определением матрицы;
- `algo` — модуль с реализациями алгоритмов умножения матриц;
- `measurement` — модуль с реализаций функции подсчета затрачиваемого времени.

3.3 Реализация алгоритмов

Далее будут представлены реализации следующих алгоритмов умножения матриц:

- Листинг 3.1 — Плавная сортировка;
- Листинг 3.2 — Сортировка перемешиванием;
- Листинг 3.3 — Сортировка Шелла.

Листинг 3.1 – Плавная сортировка

```
void sort_smooth(vector<int> &arr)
{
    int n = arr.size();

    int p = n - 1;
    int q = p;
    int r = 0;

    while (p > 0) {
        if ((r & 0x03) == 0) {
            heapify(arr, r, q);
        }

        if (leonardo(r) == p) {
            r = r + 1;
        }
        else {
            r = r - 1;
            q = q - leonardo(r);
            heapify(arr, r, q);
            q = r - 1;
            r = r + 1;
        }

        swap(arr[0], arr[p]);
        p = p - 1;
    }

    for (int i = 0; i < n - 1; i++) {
        int j = i + 1;
```

```
31          while (j > 0 && arr[j] < arr[j - 1]) {
32              swap(arr[j], arr[j - 1]);
33              j = j - 1;
34          }
35      }
36
37  }
```

Листинг 3.2 – Сортировка перемешиванием

```
1  void sort_shaker(vector<int> &arr)
2  {
3    int m = arr.size();
4     int i, j, k;
5     for(i = 0; i < m;) {
6        for(j = i+1; j < m; j++) {
7           if(arr[j] < arr[j-1])
8              swap(&arr[j], &arr[j-1]);
9        }
10       m--;
11       for(k = m-1; k > i; k--) {
12          if(arr[k] < arr[k-1])
13             swap(&arr[k], &arr[k-1]);
14       }
15       i++;
16    }
17 }
```

Листинг 3.3 – Сортировка Шелла

```
1  void sort_shell(vector<int> &arr)
2  {
3    int n = arr.size();
4      for (int gap = n/2; gap > 0; gap /= 2)
5      {
6          for (int i = gap; i < n; i += 1)
7          {
8              int temp = arr[i];
9              int j;
10             for (j = i; j >= gap && arr[j - gap] > temp; j -=
                   gap)
11                 arr[j] = arr[j - gap];
12             arr[j] = temp;
13         }
```

```
14          }
15  }
```

3.4 Функциональные тесты

В данном разделе будут представлены функциональные тесты, проверяющие работу алгоритмов сортировок.

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировок.

Таблица 3.1 – Тестирование функций

Входной массив	**Результат**	**Ожидаемый результат**
$[15, 25, 35, 45]$	$[15, 25, 35, 45]$	$[15, 25, 35, 45]$
$[55, 45, 35, 25]$	$[25, 35, 45, 55]$	$[25, 35, 45, 55]$
$[-10, -20, -30, -25]$	$[-30, -25, -20, -10]$	$[-30, -25, -20, -10]$
$[40, -10, -30, 75]$	$[-30, -10, 40, 75]$	$[-30, -10, 40, 75]$
$[100]$	$[100]$	$[100]$
$[-20]$	$[-20]$	$[-20]$
$[1.1, 2.2, 3.3, 4.4]$	$[1.1, 2.2, 3.3, 4.4]$	$[1.1, 2.2, 3.3, 4.4]$
$[1.1, -2.2, 3.3, -4.4]$	$[-4.4, -2.2, 1.1, 3.3]$	$[-4.4, -2.2, 1.1, 3.3]$
$[-1.1, -2.2, -3.3, -4.4]$	$[-4.4, 3.3, -2.2, -1.1]$	$[-4.4, -3.3, -2.2, -1.1]$
$[10, 10]$	$[10, 10]$	$[10, 10]$

Вывод

Были реализованы алгоритм плавной сортировки, алгоритм сортировки перемешиванием, алгоритм сортировки Шелла. Проведено тестирование реализованных алгортимов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

- Процессор: Intel Core i7 9750H 2.6 ГГц;
- Оперативная память: 16 ГБ;
- Операционная система: Kubuntu 22.04.3 LTS x86_64 Kernel: 6.2.0-36-generic

Во время проведения измерений времени ноутбук был подключен к сети электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 показан пример работы с программой.

```
po4ti@Po4ti-PC:~/Документы/GitHub/Lab_03_AA$ ./app.exe
1 - Плавная сортировка
2 - Сортировка перемешиванием
3 - Сортировка Шелла
0 - Выход
1
9 8 7 6 5 4 3 2 1
Отсортированный массив:
1 2 3 4 5 6 7 8 9
1 - Плавная сортировка
2 - Сортировка перемешиванием
3 - Сортировка Шелла
0 - Выход
2
5 9 8 948 4787 5 944 4
Отсортированный массив:
4 5 5 8 9 944 948 4787
1 - Плавная сортировка
2 - Сортировка перемешиванием
3 - Сортировка Шелла
0 - Выход
```

Рисунок 4.1 – Демонстрация работы программы.

4.3 Временные характеристики

Исследование временных характеристик реализованных алгоритмов производилось на массивах размером 1 – 500 с шагом 10.

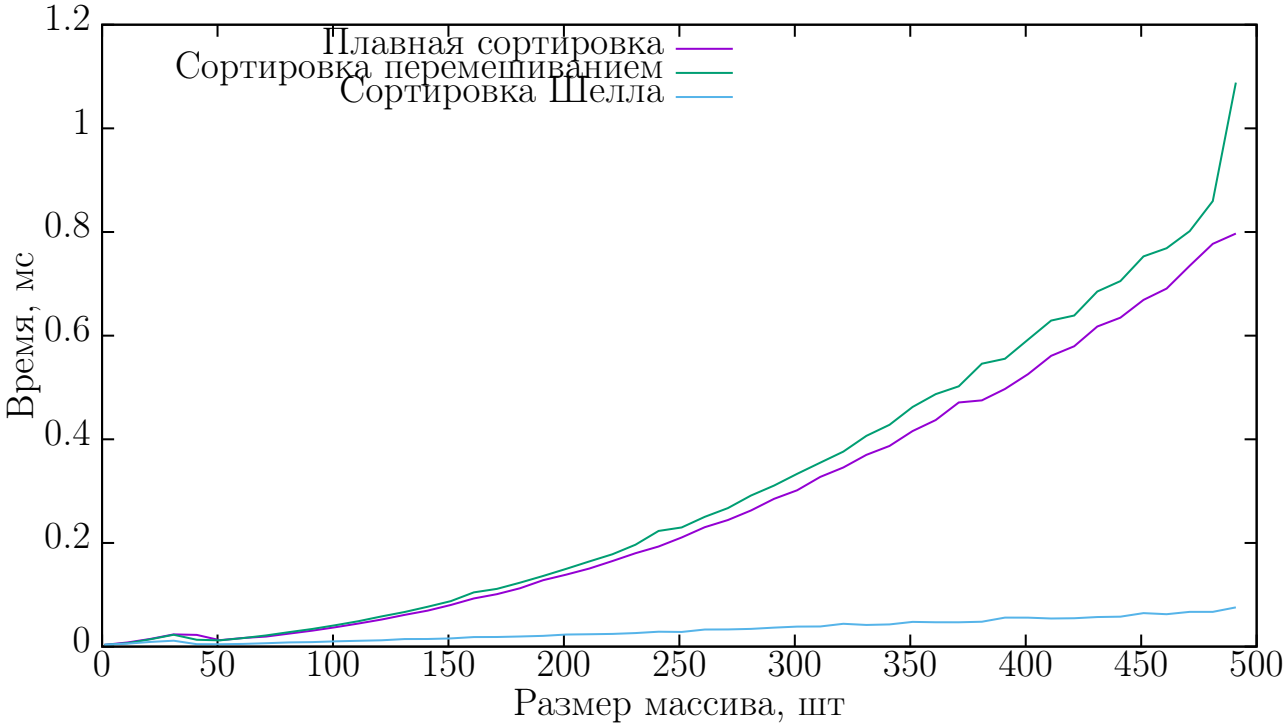

Рисунок 4.2 – Результат измерений времени работы (в мс) алгоритмов сортировок на неотсортированных массивах

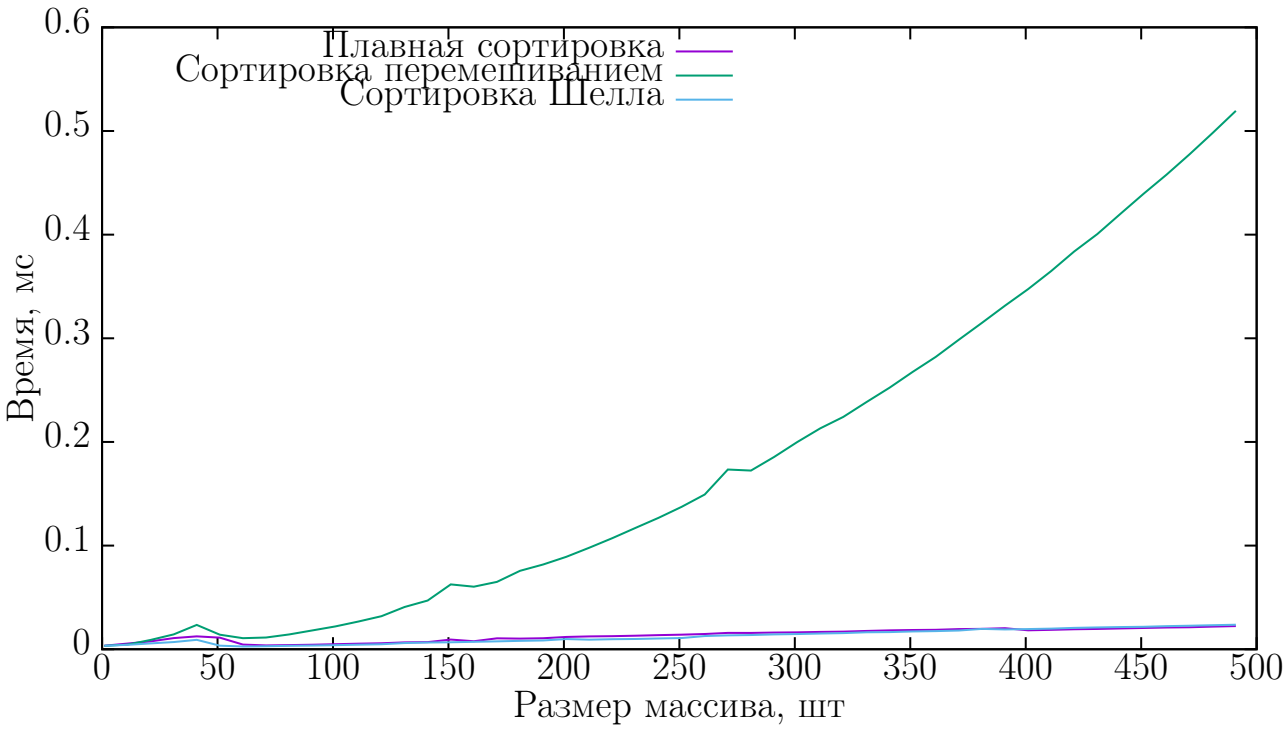

Рисунок 4.3 – Результат измерений времени работы (в мс) алгоритмов сортировок на отсортированных массивах

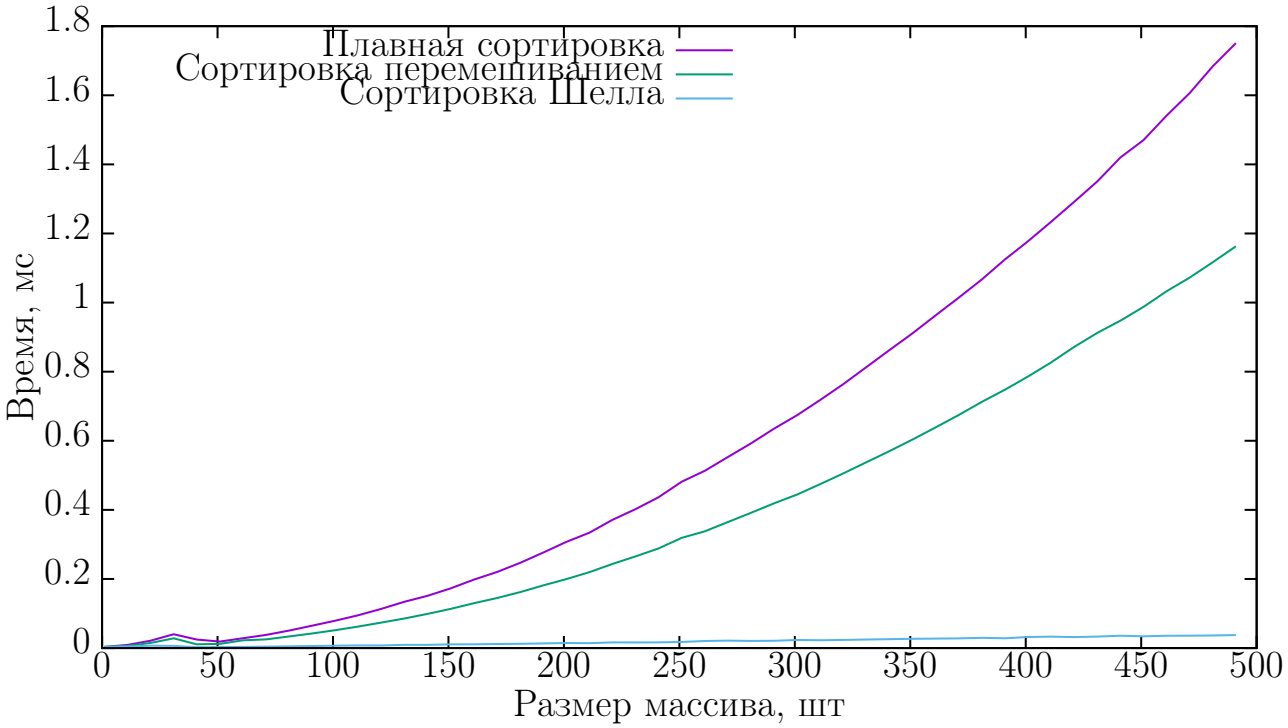

Рисунок 4.4 – Результат измерений времени работы (в мс) алгоритмов сортировок на отсортированных обратно массивах

4.4 Характеристики по памяти

Пусть дан массив N элементов сравниваемого типа int. Тогда затраты по памяти для алгоритмов будут следующие:

Плавная сортировка

Используемые переменные:

- k первых чисел леонардо – $k \cdot size(int)$
- переменные i, j, k в heapify – $3 \cdot size(int)$
- переменные p q r в теле алгоритма – $3 \cdot size(int)$
- массив – $N \cdot size(int)$
- переменная-буфер для обмена элементов местами – $1 \cdot size(int)$

Итого, для плавной сортировки:

$$M_{smooth} = k \cdot size(int) + 3 \cdot size(int) + 3 \cdot size(int) + N \cdot size(int) + 1 \cdot size \quad (4.1)$$

$$M_{smooth} = (k + N + 7) \cdot size(int) \quad (4.2)$$

Сортировка перемешиванием

Используемые переменные:

- переменные right, left – $2 \cdot size(int)$
- массив – $N \cdot size(int)$
- переменная-буфер для обмена элементов местами – $1 \cdot size(int)$

Итого, для сортировки перемешиванием:

$$M_{shake} = 2 \cdot size(int) + N \cdot size(int) + 1 \cdot size = (3 + N) \cdot size(int) \quad (4.3)$$

Сортировка Шелла

Используемые переменные:

- переменные right, left, gap – $3 \cdot size(int)$
- массив – $N \cdot size(int)$
- переменная-буфер для обмена элементов местами – $1 \cdot size(int)$

Итого, для сортировки Шелла:

$$M_{shell} = 3 \cdot size(int) + N \cdot size(int) + 1 \cdot size = (4 + N) \cdot size(int) \quad (4.4)$$

Вывод

Самым быстрым алгоритмом на неотсортированных массивах является сортировка Шелла. Самым медленным - сортировка перемешиванием.

На отсортированных массивах сортировка Шелла и плавная сортировка имеют одинаковую асимптотику и являются самыми быстрыми.

При отсортированных в обратном порядке массивах самым быстрым алгоритмом является сортировка Шелла, самым медленным - плавная сортировка.

Самым эффективным алгоритмом по затраченному объему памяти является сортировка перемешиванием. Наименее эффективным – плавная сортировка.

Заключение

По времени выполнения, алгоритм сортировки Шелла является самым эффективным, имея время выполнения быстрее других рассмотренных алгоритмов при любых типах массов.

По затрачиваемой памяти, алгоритм сортировки перемешиванием является самым эффективным, требуя наименьший объем затрачиваемой памяти.

Самыми эффективными по трудоемкости при неотсортированном массиве являются сортировки перемешиваем и Шелла, имеющие одинаковую асимптотическую сложность.

В ходе выполнения лабораторной работы были решены следующие задачи:

1. Были ознакомлены со следующими алгоритмами сортировок:
   - Плавная сортировка (smoothsort);
   - Сортировка перемешиванием;
   - Сортировка Шелла.
2. Были реализованы алгоритмы сортировок;
3. Выполнены замеры затрат реализаций алгоритмов по памяти;
4. Выполнены замеры затрат реализаций алгоритмов по процессорному времени;
5. Проведен сравнительный анализ алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сортировка перемешиванием [Электронный ресурс]. — — Режим доступа: https://alley-science.ru/domains_data/files/january-2018/ANALIZShEYKER-SORTIROVKIVMASSIVAH.pdf (дата обращения: 25.09.2022).
2. *Липачѐв. Е.* Технология программирования. Методы сортировки данных: учебное пособие. //. — — Казань: Казанский университет, 2017. — С. 59.
3. *Д.В. Шагбазян А.А. Штанюк Е. М.* Алгоритмы сортировки. Анализ, реализация, применение: учебное пособие. //. — — Нижний Новгород: Нижегородский государственный университет, 2019. — С. 22—25.
4. *Д. К.* Искусство программирования для ЭВМ. Том 3. Сортировка ипоиск. //. — — М.: ООО «И.Д. Вильямс», 2014. — С. 824.
5. Документация по Microsoft C++ [Электронный ресурс]. — — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170&viewFallbackFrom=vs-2017 (дата обращения: 25.09.2022).
6. Standard library header <ctime> [Электронный ресурс]. — — Режим доступа: https://en.cppreference.com/w/cpp/header/ctime.