Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский государственный технический университет
имени Н. Э. Баумана (национальный исследовательский университет»)

Курс: «Анализ алгоритмов»

Лабораторная работа №3

Тема работы:
«Сортировка массивов»

Студент: Волков Е. А.
Преподаватели: Волкова Л. Л.
Строганов Ю. В.
Группа: ИУ7-55Б

2019

Содержание

Введение

Цель лабораторной работы: изучение метода динамического программирования на материале алгоритмов сортировок массивов, а именно: на сортировке пузырьком, сортировке слиянием и быстрой сортировке.

Задачи работы:

1) изучение алгоритмов сортировок;
2) применение метода динамического программирования для реализации указанных алгоритмов;
3) практические навыки реализации указанных алгоритмов: сортировки пузырьком, сортировки слиянием и быстрой сортировки;
5) сравнительный анализ алгоритмов по затрачиваемым ресурсам (времени);
4) экспериментальное подтверждение различий во временно́й эффективности алгоритмов при помощи разработанного программного обеспечения на материале замеров процессорного времени на варьирующихся размерностях матриц;
5) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитический раздел

В данном разделе будут рассмотрены алгоритмы сортировки массивов.

1.1 Описание алгоритмов

Сортировка - процесс перегруппировки элементов массива в некотором определенном порядке. Сортировка предпринимается для того, чтобы облегчить последующий поиск элементов в отсортированном массиве [3].

Алгоритмы сортировки активно применяются во всех областях, которым необходимо хранить последовательность чисел, таких как:

1) математика

2) физика

3) экономика

4) и т. д.

1.1.1 Алгоритм сортировки пузырьком

Сортировка пузырьком - один из самых простых алгоритмов сортировки. Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма) [4].

1.1.2 Алгоритм быстрой сортировки

Быстрая сортировка — рекурсивный алгоритм сортировки. Быстрая сортировка выбирает элемент списка, называемый осевым, а затем переупорядочивает список таким образом, что все элементы, меньшие осевого, оказываются перед ним, а большие элементы –за ним. В каждой из частей списка элементы не упорядочиваются. Если R — окончательное положение осевого элемента, то нам известно лишь, что все значения в позициях с первой по R — 1 меньше осевого,

а значения с номерами от R + 1 до N больше осевого. Затем алгоритм вызывается рекурсивно на каждой из двух частей. При вызове процедуры Quicksort на списке, состоящем из одного элемента, он ничего не делает, поскольку одноэлементный список уже отсортирован [1].

1.1.3 Алгоритм сортировки слиянием

Сортировка слиянием — еще одна из рекурсивных сортировок. В ее основе лежит замечание, согласно которому слияние двух отсортированных списков выполняется быстро. Сортировку слиянием можно записать в виде рекурсивного алгоритма, выполняющего работу, двигаясь вверх по рекурсии. При взгляде на нижеследующий алгоритм видно, что он разбивает список пополам до тех пор, пока номер первого элемента куска меньше номера последнего элемента в нем. Если же в очередном куске это условие не выполняется, это означает, что мы добрались до списка из одного элемента, который тем самым уже отсортирован. После возврата из двух вызовов процедуры MergeSort со списками длиной один вызывается процедура которая сливает эти два списка, в результате чего получается отсортированный список длины два. При возврате на следующий уровень два списка длины два сливаются в один отсортированный список длины 4. Этот процесс продолжается, пока мы не доберемся до исходного вызова, при котором две отсортированные половины списка сливаются в общий отсортированный список.

Вывод

В данном разделе были рассмотрены алгоритм сортировки пузырьком (итеративный алгоритм), быстрая сортировка и сортировка слиянием (рекурсивные алгоритмы). Все они имеют разные подходы к упорядочиванию массива: пузырек перебирает элементы попарно, быстрая сортировка тоже использует перебор, но отталкивается от определенного осевого элемента, а сортировка слиянием использует принцип «разделяй и властвуй», разбивая массив на подмассивы, пока те не станут длины 1, и затем «собирает» их в нужном порядке.

2 Конструкторский раздел

В данном разделе содержатся схемы сортировок и сравнительный анализ их трудоемкости.

2.1 Разработка алгоритмов

На рис. 1, 2, 3, 4 и 5 приведены схемы сортировки пузырьком, сортировки слиянием и быстрой сортировки.

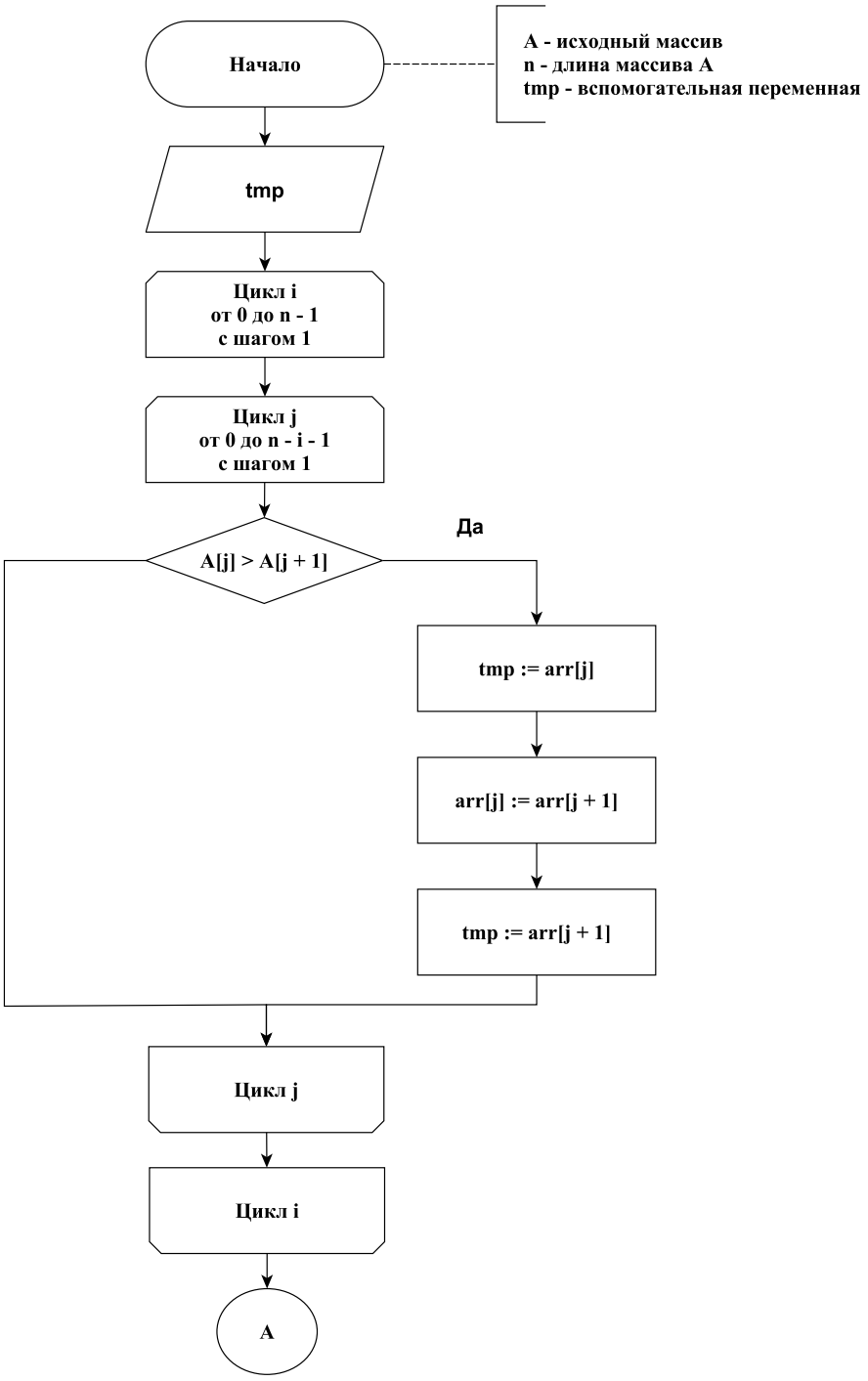

Рис. 1: Схема алгоритма сортировки пузырьком

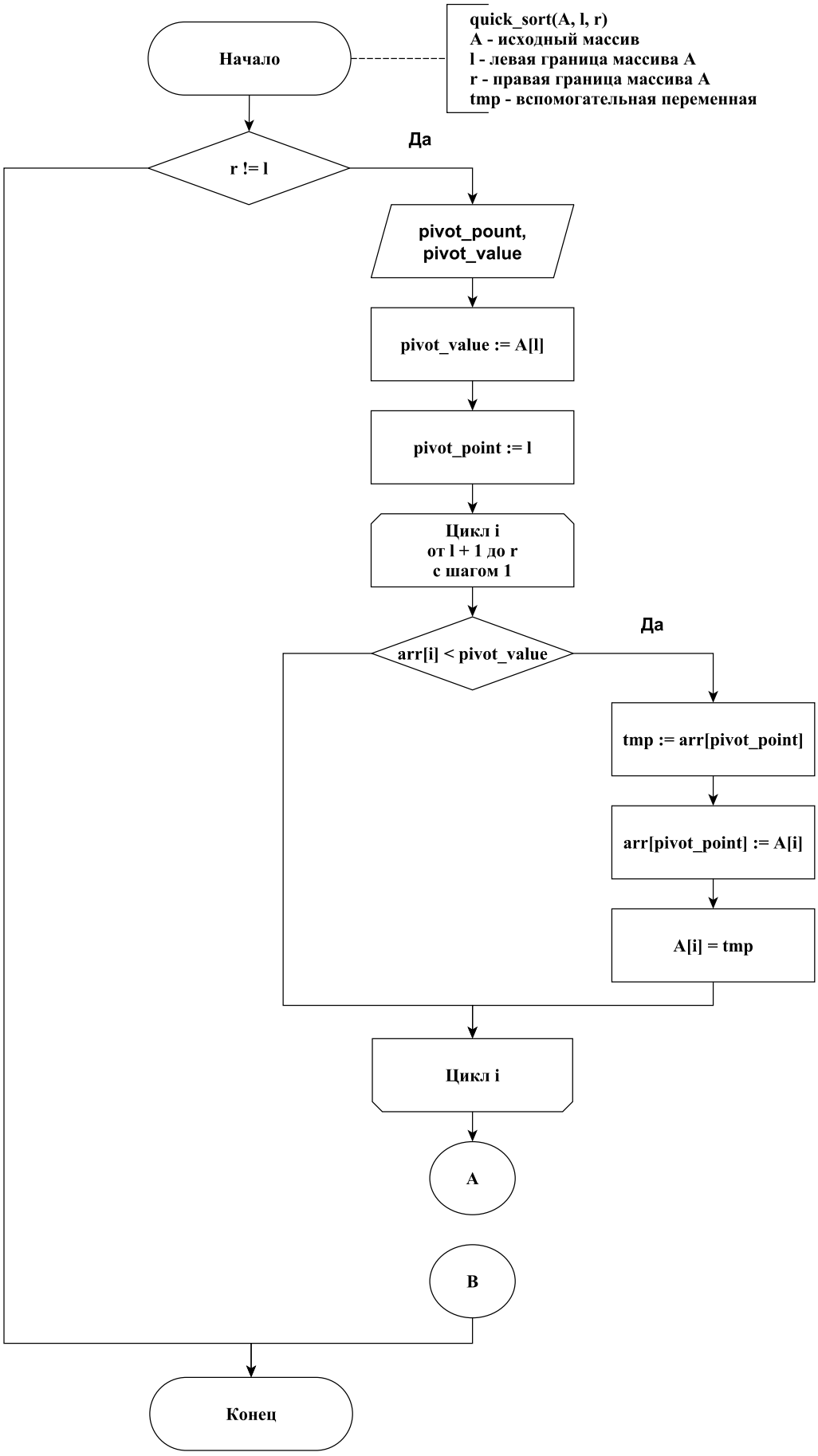

Рис. 2: Схема алгоритма быстрой сортировки (часть 1)

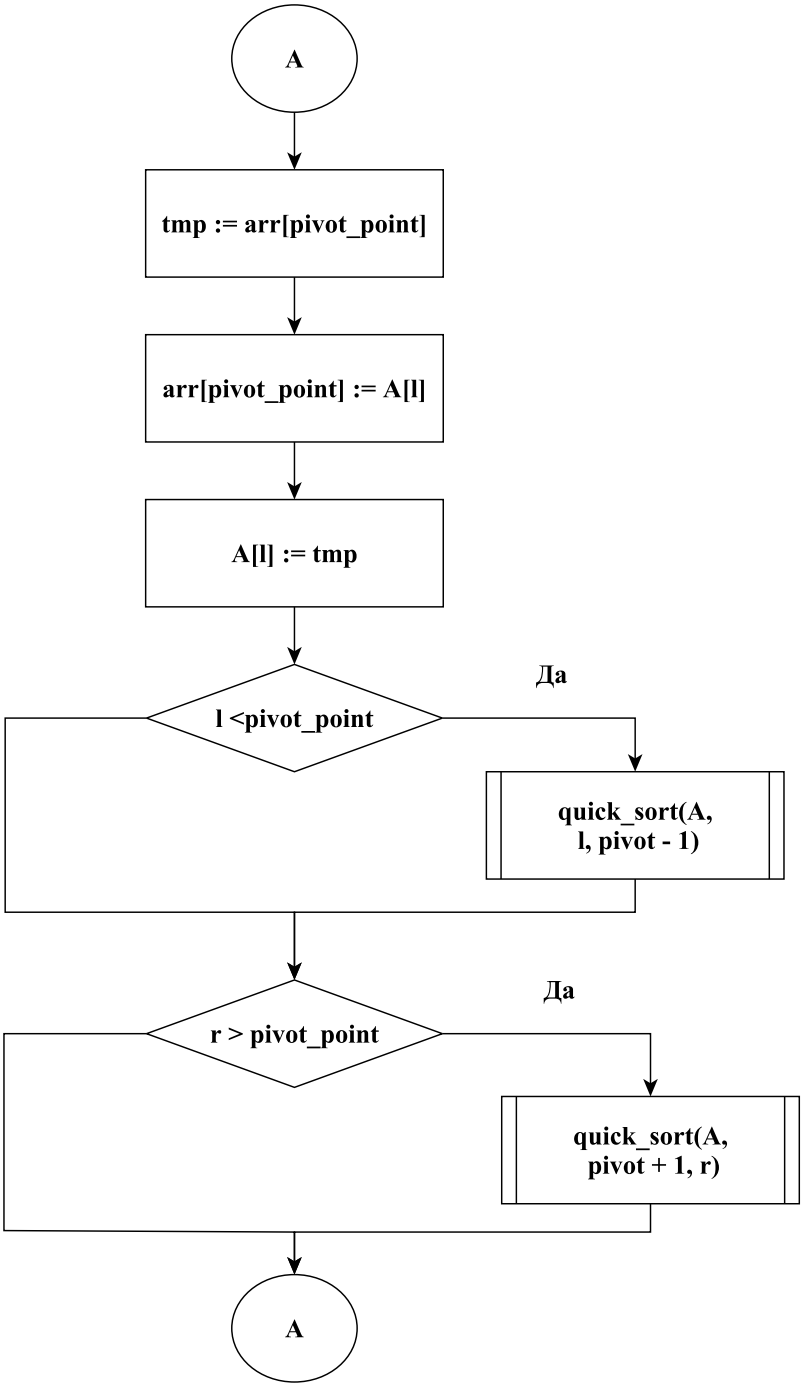

Рис. 3: Схема алгоритма быстрой сортировки (часть 2)

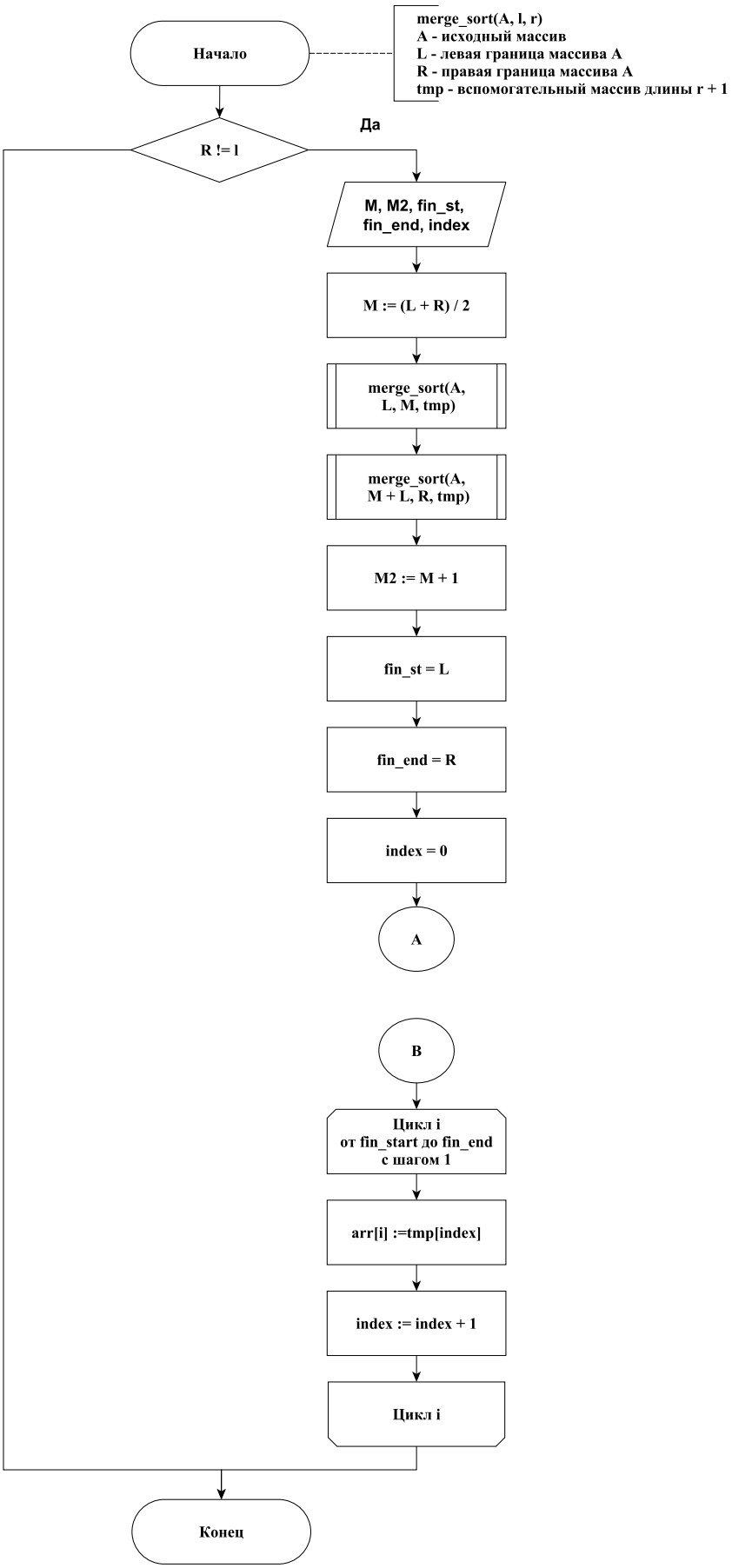

Рис. 4: Схема алгоритма сортировки слиянием (часть 1)

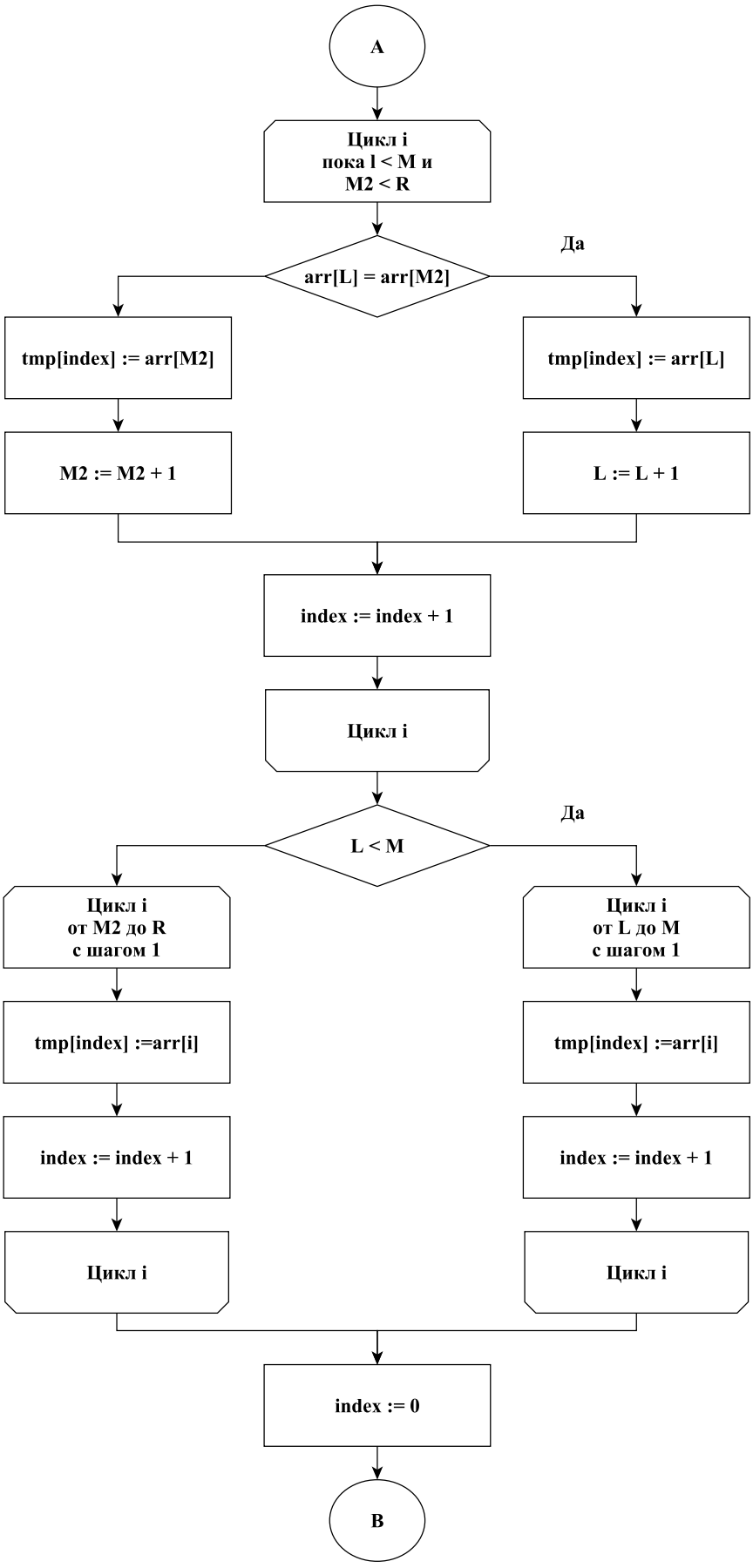

Рис. 5: Схема алгоритма сортировки слиянием (часть 2)

2.2 Сравнительный анализ алгоритмов

Трудоемкость алгоритмов измеряется в количестве необходимых операций.

Введем модель вычислений:

1) Базовые операции стоимостью 1: +, -, /, *, =, ==, <=, >= !=, +=, -=, *=, /=, [ ].

2) Стоимость цикла:

$$f_{\text{цикла}} = f_{\text{иниц}} + f_{\text{сравн}} + N(f_{\text{тела}} + f_{\text{инкр}} + f_{\text{сравн}}) = 2 + N(f_{\text{тела}} + 2)$$

3) Стоимость условного перехода примем за 0, стоимость вычисления условия остается

Найдем вычислительную сложность алгоритмов для массива длиной N элементов.

2.2.1 Сортировка пузырьком

В общем виде формула количества затрачиваемых операций выглядит так:

$$f = \underbrace{2}_{=} + 2 + N\left(2 + 2 + \frac{N}{2}\left(2 + \underbrace{4}_{if} + \begin{bmatrix} 0, \text{если требуется обмен элементов} \\ \underbrace{4}_{[\,]} + \underbrace{3}_{=} + \underbrace{2}_{+}, \text{иначе.} \end{bmatrix}\right)\right),$$

Лучшим случаем для алгоритма пузырька является уже упорядоченный массив, когда не будет происходить обмен элементов и сложность составит:

$$f = 5 + N(4 + \frac{N}{2} \times (6 + 0)) = 3NN + 4N + 5.$$

Худшим, соответственно, будет случай когда массив отсортирован в обратном порядке, тогда на каждой итерации внутреннего цикла понадобится обмен и сложность будет:

$$f = 5 + N(4 + \frac{N}{2} \times (6 + 9)) = \frac{15}{2}NN + 4N + 5.$$

Тогда для среднего случая сложность будет:

$$f = \frac{3 + \frac{15}{2}}{2}NN + 4N + 5 = \frac{21}{4}NN + 4N + 5.$$

Отсюда видно, что при любом случае сложность алгоритма $O(N^2)$

2.2.2 Сортировка слиянием

Оценим сначала трудоемкость слияния двух отсортированных массивов длиной N_A и N_B. Лучшим случаем будет ситуация, когда все элементы одного больше (меньше) всех элементов другого. В таком случае, мы переносим в результирующий массив сначала все элементы одного массива, затем другого при минимальном количестве сравнений. Пусть Все элементы массива N_A меньше всех элементов массива N_B, тогда:

$$f(N_A, N_B) = \underbrace{4}_{=} + \underbrace{1}_{+} + 2 + N_A(2 + \underbrace{4}_{[]} + \underbrace{2}_{=} + \underbrace{1}_{+} + \underbrace{1}_{<}) + \underbrace{1}_{<} + 2 +$$

$$+ N_B(3 + \underbrace{2}_{[]} + \underbrace{1}_{=}) + \underbrace{1}_{=} + 2 + (N_A + N_B)(3 + \underbrace{2}_{[]} + \underbrace{1}_{=}) =$$

$$= 16N_A + 12N_B + 13.$$

Худшим случаем является случай, когда значения массив N_A и N_B идут «через один», например 1-й элемент N_A меньше 1-го элемента N_B, 1-й элемент N_B меньше 2-го элемента N_A и т. д. В таком случае сложность будет:

$$f(N_A, N_B) = \underbrace{4}_{=} + \underbrace{1}_{+} + 2 + (N_A + (2 + \underbrace{4}_{[]} + \underbrace{2}_{=} + \underbrace{1}_{+} + \underbrace{1}_{<}) + \underbrace{1}_{<} + 2$$

$$+ (N_A + N_B)(3 + \underbrace{2}_{[]} + \underbrace{1}_{=}) =$$

$$= 16N_A + 16N_B + 10.$$

Теперь оценим саму сортировку. На каждой итерации происходит разбиение массива длины N на два длины $\frac{N}{2}$. Для их слияния понадобится в худшем случае $16N + 10$ операций и $14N + 10$ операций в худшем случае. Получаем рекуррентные соотношения на сложность в наихудшем (W) и наилучшем (B) случаях:

$$W(N) = 2W(16N) + 13,$$

$$W(0) = W(1) = 0;$$

$$B(N) = 2B(14N) + 10,$$

$$B(0) - 5(1) = 0.$$

Решив эти рекуррентные соотношения, получим значения близкие к приведенным в книге Дж. Макконнелла «Анализ алгоритмов. Активный обучающий подход» для худшего случая:

$$f = N \log_2 N - N + 1.$$

Для лучшего случая [1]:

$$f = \frac{N \log_2 N}{2}.$$

Отбросив константное значение видим, что при любом случае алгоритм имеет асимптотику $O(N \log N)$

2.2.3 Быстрая сортировка

Худшим случаем для быстрой сортировки является отсортированный в каком-либо направлении массив, т. к. в этом случае в каждом рекурсивном вызове массив разбивается на два других, один из которых пуст, а другой целиком содержит укороченный на 1 элемент исходный массив. Таким образом сложность для худшего случая:

$$f = \underbrace{2}_{=} + \underbrace{1}_{+} + \frac{N}{2}(\underbrace{2}_{>} + \underbrace{3}_{=} + \underbrace{1}_{==} + \underbrace{1}_{[]} + 3 +$$

$$+ N(2 + \underbrace{1}_{=} + \underbrace{1}_{[]} + \underbrace{1}_{<} + \underbrace{9}_{swap}) + \underbrace{9}_{swap}) =$$

$$= 7NN + 5N + 12$$

В среднем случае, когда элементы массива достаточно перемешаны, скорость работы алгоритма определяется следующим рекуррентным соотношением [1]:

$$A(N) = (N-1) + \frac{1}{N}\left(2\sum_{i=1}^{N-1} A_i\right) \text{ для N} > 2$$

$$A(1) = A(0) = 1$$

Проанализировав их, получим:

$$f \approx 1.4(N+1)\log_2 N$$

Таким образом, в худшем случае быстрая сортировка имеет сложность $O(N^2)$, а в среднем $N \log N$

Вывод

В данном разделе были представлены схемы сортировки пузырьком, сортировки слиянием и быстрой сортировки. Был произведен аналитический анализ трудоемкости сортировок, в ходе которого выяснилось, что самой эффективной является сортировка слиянием, имеюшая сложность $O(N \log_2 N)$ на любом массиве. Быстрая сортировка в среднем случае также имеет сложность

$O(N \log_2 N)$, но в худшем случае (когда массив изначально отсортирован в прямом или обратном порядке) имеет сложность $O(N^2)$. Сортировка пузырьком является самой медленной - при любых исходных данных она имеет сложность $O(N^2)$.

3 Технологический раздел

В данном разделе будут описаны требования к программному обеспечению и средства реализации, приведен листинг программы и выполнен сравнительный анализ трудоемкости алгоритмов.

3.1 Требования к программному обеспечению

Входные данные:

1) целое положительное число - размерность массива: N;
2) массив, заполненный вещественными числами.

Выходные данные: исходный массив, отсортированный по возрастанию.

На рис. 6 приведена функциональная схема сортировки массива.

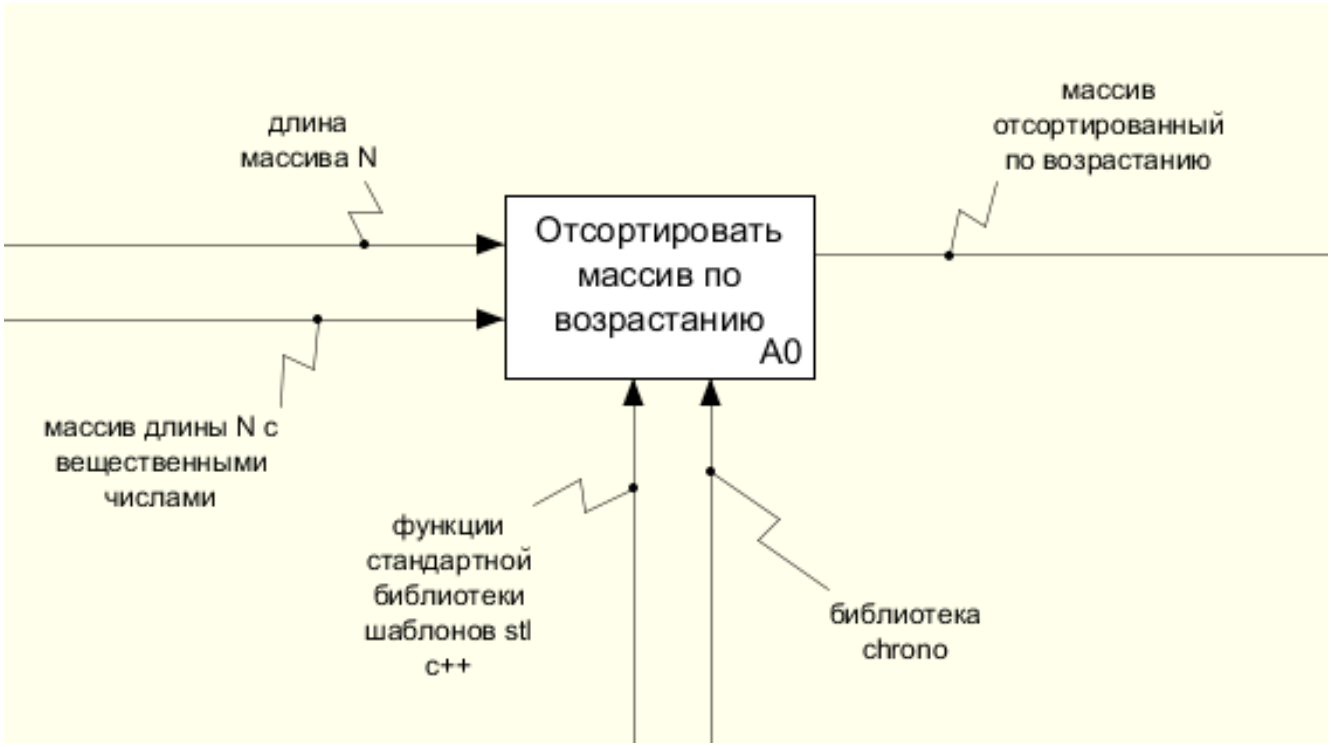

Рис. 6: Функциональная схема сортировки массива

3.2 Средства реализации

Программа написана на языке C++, т. к. этот язык предоставляет программисту широкие возможности реализации самых разнообразных алгоритмов, обладает высокой эффективностью и значительным набором стандартных классов и процедур. В качестве среды разработки использовался фреймворк QT 5.13.1.

Для обработки массивов был использован стандартный контейнерный класс std::vector.

Для замера времени выполнения программы использовалась библиотека chrono.

3.3 Листинг программы

В листингах 1, 2 и 3 представлены функции сортировки пузырьком, сортировки слиянием и быстрой сортировки.

Листинг 1: Сортировка пузырьком

```
void bubble_sort(Vector &arr) {
    size_t n = arr.size();
    double tmp = 0;
    for (size_t i = 0; i < n - 1; i++) {
        for (size_t j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                std::swap(arr[j], arr[j + 1]);
            }
        }
    }
}
```

Листинг 2: Быстрая сортировка

```
void quick_sort(Vector &arr, size_t l, size_t r) {
    if (r == l) return;
    size_t pivot = pivot_arr(arr, l, r);
    if (l < pivot) {
        quick_sort(arr, l, pivot - 1);
    }
    if (r > pivot) {
        quick_sort(arr, pivot + 1, r);
    }
}

size_t pivot_arr(Vector &arr, size_t l, size_t r) {

    double pivot_value = arr[l];
    size_t pivot_point = l;
    for (size_t i = l + 1; i <= r; i++) {
        if (arr[i] < pivot_value) {
            pivot_point++;
            std::swap(arr[pivot_point], arr[i]);
        }
    }
    std::swap(arr[l], arr[pivot_point]);
    return pivot_point;

}
```

Листинг 3: Сортировка слиянием

```
void merge_sort(Vector &arr, size_t l, size_t r, Vector &tmp) {
    if (l == r) return;
    size_t m = (l + r) / 2;
    merge_sort(arr, l, m, tmp);
    merge_sort(arr, m + 1, r, tmp);
    merge(arr, l, m, r, tmp);
}

void merge(Vector &arr, size_t st1, size_t end1, size_t end2, Vector &tmp) {
    size_t st2 = end1 + 1;
    size_t fin_st = st1;
    size_t fin_end = end2;
    size_t index = 0;
    while (st1 <= end1 && st2 <= end2) {
        if (arr[st1] < arr[st2]) {
            tmp[index] = arr[st1];
            st1++;
        }
        else {
            tmp[index] = arr[st2];
            st2++;
        }
        index++;
    }

    if (st1 <= end1) {
        for (size_t i = st1; i <= end1; i++, index++) {
            tmp[index] = arr[i];
        }
    }
    else {
        for (size_t i = st2; i <= end2; i++, index++) {
            tmp[index] = arr[i];
        }
    }

    index = 0;
    for (size_t i = fin_st; i <= fin_end; i++, index++) {
        arr[i] = tmp[index];
    }


}
```

3.4 Тестовые данные

Программа должна корректно сортировать массив при следующих входных данных:

Таблица 1: Тестовые данные

№	Исходный массив	Отсортированный массив
1	{1}	{1}
2	{2, 1}	{1, 2}
3	{1, 2, 3, 4, 5}	{1, 2, 3, 4, 5}
4	{5, 4, 3, 2, 1}	{1, 2, 3, 4, 5}
5	{2, 2, 2}	{2, 2, 2}
6	{5, 2, 3, 4, 1}	{1, 2, 3, 4, 5}
7	{-5, -4, -3, -2, -1}	{-5, -4, -3, -2, -1}
8	{-5, 4, -3, 2, -1, 0}	{-5, -3, -1, 0, 2, 4}
9	{1, 1, 3, 6, 1, 5, 9, 3}	{1, 1, 1, 3, 3, 5, 6, 9}
10	{3.98, 0, 4.1, -2.34, 10.876, 3.99, -8}	{-8, -2.34, 0, 3.98, 3.99, 4.1, 10.876}
11	{1000, -2000, 3000, 0, 5000, -9000, -5000}	{-9000, -5000, -2000, 0, 1000, 3000, 5000}

Все тесты успешно пройдены.

Вывод

В данном разделе были рассмотрены требования к программному обеспечению, в качестве средств реализации выбраны язык C++ и фреймворк QT версии 5.13.1, приведён листинг программы и тестовые данные. Все тесты успешно прошли тестирование, результаты совпали с ожидаемыми.

4 Исследовательский раздел

4.1 Примеры работы

На рис. 7, 8, 9, 10 и 11 приведены примеры работы программы для различных входных данных.

```
Enter number of array: -5
Incorrect input. Try again: dsfsg
Incorrect input. Try again: 5
matr[0] = g
Incorrect input. Try again: -3
matr[1] = 4.3
matr[2] = 0
matr[3] = -1
matr[4] = 3
Source array:-3 4 0 -1 3
Result of bubble sort:
-3 -1 0 3 4
Result of quick sort:
-3 -1 0 3 4
Result of merge sort:
-3 -1 0 3 4
```

Рис. 7: Пример работы программы для некорректного ввода

```
Enter number of array: 5
matr[0] = 1
matr[1] = 2
matr[2] = 3
matr[3] = 4
matr[4] = 5
Source array:1 2 3 4 5
Result of bubble sort:
1 2 3 4 5
Result of quick sort:
1 2 3 4 5
Result of merge sort:
1 2 3 4 5
```

Рис. 8: Пример работы программы для отсортированного массива

```
Enter number of array: 6
matr[0] = 6
matr[1] = 5
matr[2] = 4
matr[3] = 3
matr[4] = 2
matr[5] = 1
Source array:6 5 4 3 2 1
Result of bubble sort:
1 2 3 4 5 6
Result of quick sort:
1 2 3 4 5 6
Result of merge sort:
1 2 3 4 5 6
```

Рис. 9: Пример работы программы для отсортированного в обратном порядке массива

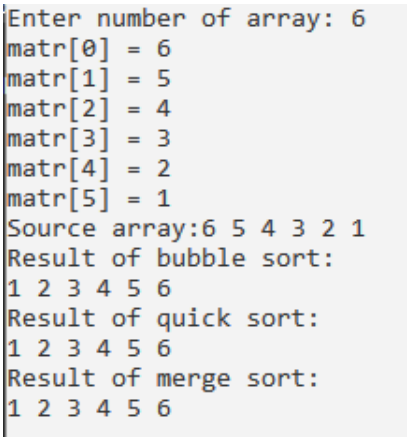

Рис. 10: Пример работы программы для неупорядоченного массива

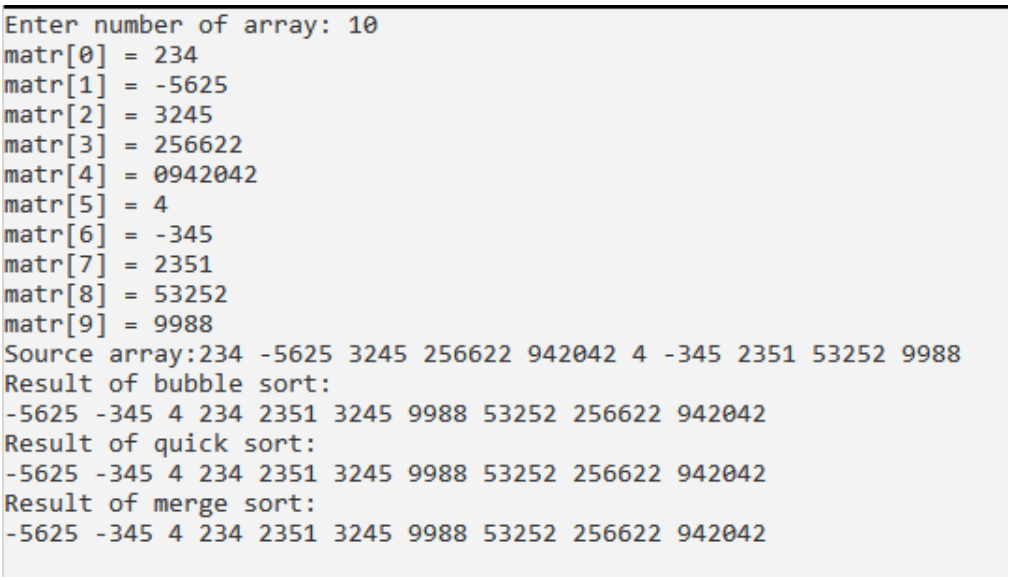

Рис. 11: Пример работы программы для массива с большими значениями

4.2 Постановка эксперимента

Необходимо выполнить следующие замеры времени:

1. Сравнить время сортировок алгоритмов на массивах с произвольно сгенерированными числами.

2. Провести аналогичное сравнение на массивах, с самыми подходящими данными для конкретных алгоритмов (лучший случай). Для сортировки пузырьком это уже отсортированный массив, для быстрой сортировки - произвольно сгенерированный, для сортировке слиянием принципиальной разницы нет.

3. Провести аналогичное сравнение на массивах, с самыми неподходящими данными для конкретных алгоритмов (худший случай). Для сортировки пузырьком это массив, отсортированный в обратном порядке, для быстрой сортировки - отсортированный в любом направлении массив, для сортировке слиянием принципиальной разницы нет.

Все замеры произвести на массивах длиной от 100 до 1000 элементов с шагом 100, каждый замер выполняется 10000 раз.

4.3 Сравнительный анализ на материале экспериментальных данных

На рис. 12 приводятся графики времени сортировки алгоритмов на массивах с произвольно сгенерированными числами.

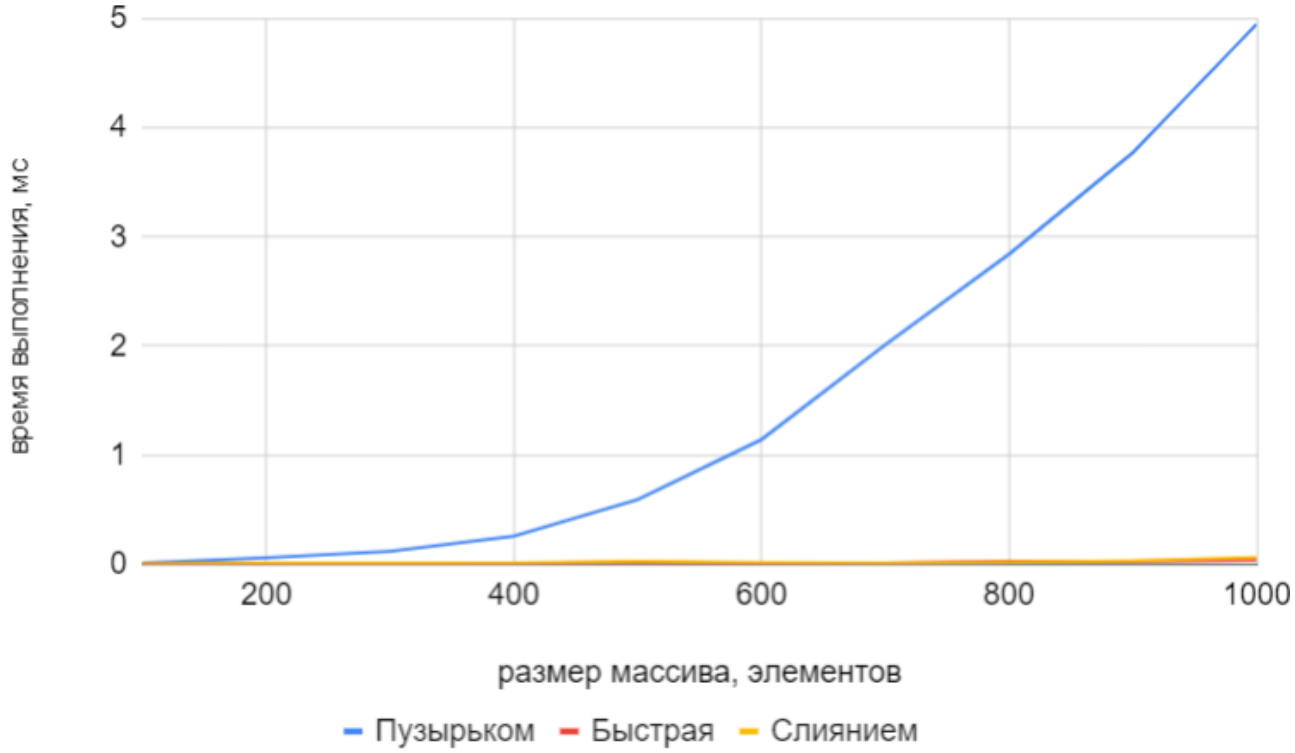

Рис. 12: Время выполнения алгоритмов на массивах с произвольно сгенерированными числами

Как видно, сортировка пузырьком существенно проигрывает быстрой сортировке и сортировке слиянием. Графики соответствуют посчитанным в разделе 2.2 асимптотикам для случайного массива: $O(N^2)$ для пузырьковой сортировки и $O(N \log N)$ для быстрой сортировки и сортировки слиянием.

На рис. 13 приводятся графики времени сортировки алгоритмов при лучшем случаи.

Как видно, пузырек стал выполняться быстрее (1.5 мс на 1000 элементов, по сравнению с прежними 5-ю мс), характер зависимости при этом остался таким же как у замеров с произвольно заполненными массивами. Быстрая сортировка и сортировка слиянием сохранили свою асимптотику в $O(N \log N)$.

На рис. 14 приводятся графики времени сортировки алгоритмов при худшем случае.

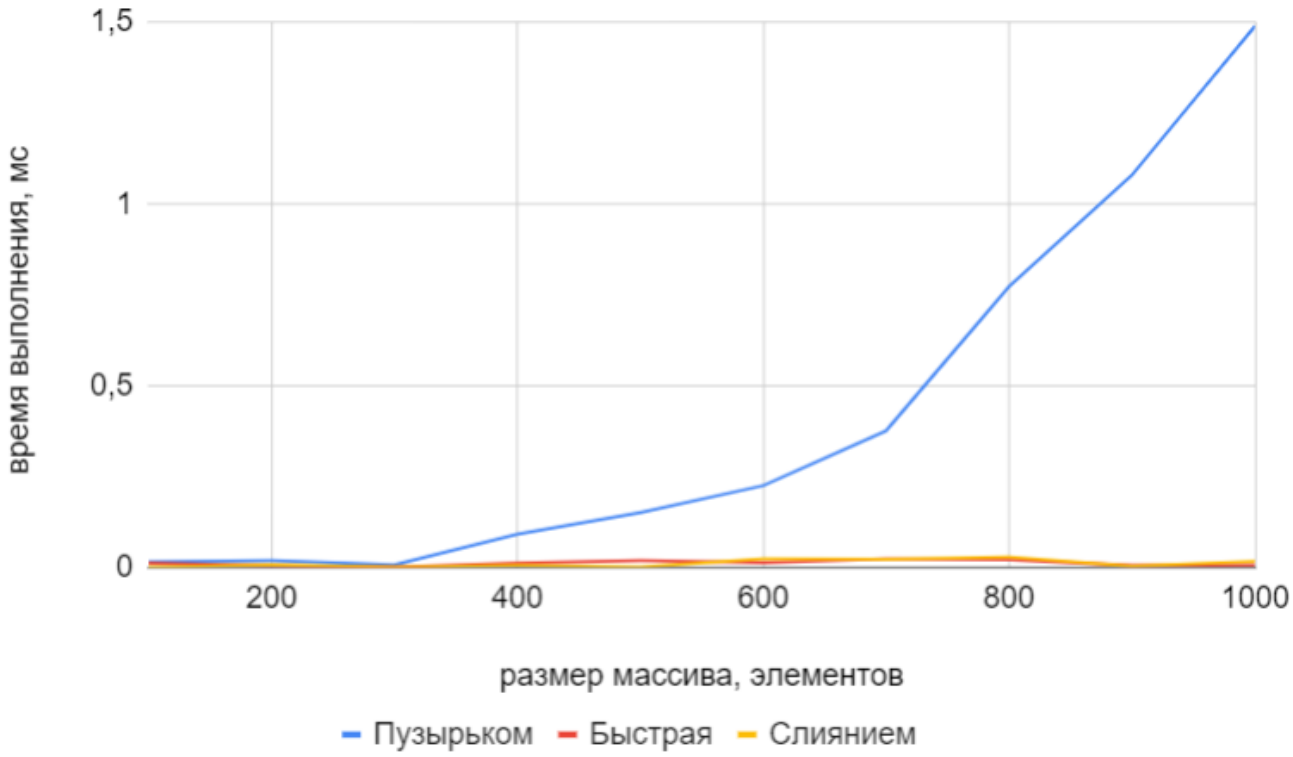

Рис. 13: Время выполнения алгоритмов при худшем случае

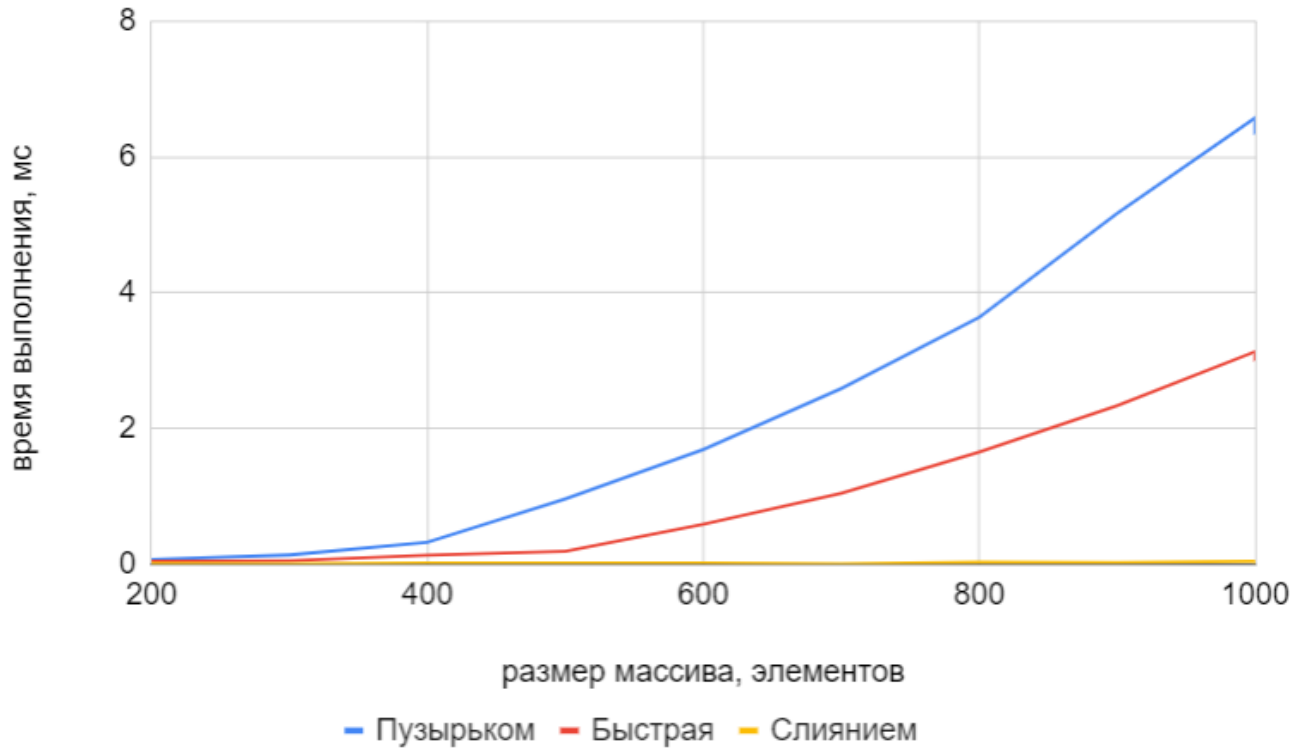

Рис. 14: Время выполнения алгоритмов при лучшем случае

Видно, что в данном случае быстрая сортировка существенно замедлилась, что соответствует ее теоретической оценке для худшего случая в $O(N^2)$. Пузырек также упал в производительности по сравнению с замерами для среднего случая (7 секунд для массива в 1000 элементов, по сравнению с 5 секунда-

ми для замера с произвольными числами). Сортировка слиянием по-прежнему остается самой производительной.

Вывод

В результате тестирования программой были успешно пройдены все заявленные тесты. Эксперименты замера времени показали, что самой эффективной является сортировка слиянием, которая вне зависимости от входных данных имеет сложность $O(N \log N)$. Однако ее недостатком является необходимость выделять память под вспомогательный массив. Быстрая сортировка в среднем случае также показывает сложность $O(N \log N)$, но в худшем случае (например, при отсортированном массиве) опускается до $O(N^2)$. Пузырек оказался самым неэффективным - его сложность в любом случае $O(N^2)$.

Заключение

В ходе работы было выполнено следующее:

1) изучены алгоритмы сортировок;

2) применены методы динамического программирования для реализации указанных алгоритмов;

3) получены практические навыки реализации указанных алгоритмов: сортировки пузырьком, сортировки слиянием и быстрой сортировки;

5) проведен сравнительный анализ алгоритмов по затрачиваемым ресурсам (времени);

4) выполнено экспериментальное подтверждение различий во временно́й эффективности алгоритмов при помощи разработанного программного обеспечения на материале замеров процессорного времени на варьирующихся размерностях матриц;

5) описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

Аналитический анализ трудоемкости алгоритмов показал, что что самой эффективной является сортировка слиянием, имеюшая сложность $O(N \log N)$ на любом массиве. Быстрая сортировка в среднем случае также имеет сложность $O(N \log N)$, но в худшем случае (когда массив изначально отсортирован в прямом или обратном порядке) имеет сложность $O(N^2)$. Сортировка пузырьком является самой медленной - при любых исходных данных она имеет сложность $O(N^2)$.

Экспериментальный анализ эффективности алгоритмов подтвердил теоретическую оценку. Пузырек был стабильно медленным на любых данных(в 100-150 раз медленнее других массивов на массиве длиной 1000 элементов в среднем и лучшем случаях). Слияние давало стабильный выигрыш на любых массивах). Быстрая сортировка в лучшем и среднем случаях была эффективна как и сортировка слиянием, а в худшем случае шла наравне с пузырьком.

Список литературы

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.-М.:Техносфера, 2009.

[2] Описание алгоритмов сортировки и сравнение их производительности [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/335920/, свободный – (22.10.2019)

[3] Что такое сортировка? [Электронный ресурс]. – Режим доступа: https://dic.academic.ru/dic.nsf, свободный – (21.10.2019)

[4] Пузырьковая сортировка (Bubble-sort) [Электронный ресурс]. – Режим доступа: https://forkettle.ru/vidioteka/programmirovanie-i-set/108-algoritmy-i-struktury-dannykh/sortirovka-i-poisk-dlya-chajnikov/1004-puzyrkovaya-sortirovka, свободный – (21.10.2019)