ГУАП

КАФЕДРА № 14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		П. В. Шпигун
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 2

СОРТИРОВКА ОДНОМЕРНЫХ МАССИВОВ

по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	1446	19.12.2025	А. С. Пырву
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

Цель работы: Изучить основные алгоритмы сортировки массивов и освоить их на практике. Проверить работу алгоритмов на различных наборах данных.

Постановка задачи:

1. Определить массив, элементы которого будут упорядочиваться. Тип массива выбрать самостоятельно.

2. Разработать функции сортировки массива методами, выбранными по таблице в соответствии с вариантом задания.

3. Вызывающая программа (main) должна в диалоге выбирать метод сортировки (один из двух) и способ задания исходных данных для тестирования: длину массива и способ его формирования – ввод с клавиатуры, генерация случайных чисел или чтение из файла.

4. Выполнить сортировку массива первым алгоритмом и проконтролировать ее результат. Проверить все варианты исходного заполнения массива. Убедиться в правильности сортировки во всех случаях. Сделать выводы.

5. Повторить пункты 3 и 4 для второго алгоритма сортировки.

Формализация:

Реализованные функции:
quicksort - Реализация алгоритма быстрой сортировки
mergeArrays - Слияние двух отсортированных массивов
mergeSort - Реализация алгоритма сортировки слиянием
generate_random_array - Генерация случайного массива

Использованные библиотеки:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

Листинг программы:

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <locale.h>

// Функция обмена элементов
void swap(int* a, int* b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

// Функция разделения для быстрой сортировки
int split(int* arr, int left, int right)
{
    int center_element = arr[left];
```

```
    int i = left + 1;
    int j = right;

    while (i <= j)
    {
        while (i <= j && arr[i] <= center_element)
            i++;

        while (i <= j && arr[j] >= center_element)
            j--;

        if (i < j)
            swap(arr + i, arr + j);
    }

    swap(arr + left, arr + j);
    return j;
}

// Быстрая сортировка
void quicksort(int* arr, int left, int right)
{
    if (left >= right)
        return;

    int center_index = split(arr, left, right);
    quicksort(arr, left, center_index - 1);
    quicksort(arr, center_index + 1, right);
}

// Слияние двух отсортированных массивов
void mergeArrays(int* arr, int size_left, int size_right)
{
    int* arr_left = malloc(sizeof(int) * size_left);
    int* arr_right = malloc(sizeof(int) * size_right);

    memcpy(arr_left, arr, sizeof(int) * size_left);
    memcpy(arr_right, arr + size_left, sizeof(int) * size_right);

    int i = 0;
    int j = 0;

    while (i < size_left && j < size_right)
    {
        if (arr_left[i] < arr_right[j])
        {
            arr[i + j] = arr_left[i];
            i++;
        }
        else
        {
            arr[i + j] = arr_right[j];
            j++;
        }
    }

    while (i < size_left)
    {
        arr[i + j] = arr_left[i];
        i++;
    }

    while (j < size_right)
    {
        arr[i + j] = arr_right[j];
```

```
        j++;
    }

    free(arr_left);
    free(arr_right);
}

// Сортировка слиянием
void mergeSort(int* arr, int size)
{
    if (size <= 1)
        return;
    else if (size == 2)
    {
        if (arr[0] > arr[1])
            swap(arr, arr + 1);
        return;
    }

    int size_left = size / 2;
    int size_right = size - size_left;

    mergeSort(arr, size_left);
    mergeSort(arr + size_left, size_right);
    mergeArrays(arr, size_left, size_right);
}

// Вывод массива
void print_array(int* arr, int size)
{
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Выбор метода сортировки
int select_sort_method()
{
    int choice;
    printf("\nВыберите метод сортировки:\n");
    printf("1. Быстрая сортировка\n");
    printf("2. Сортировка слиянием\n");
    printf("Ваш выбор: ");
    scanf("%d", &choice);
    return choice;
}

// Выбор способа формирования массива
int select_input_method()
{
    int choice;
    printf("\nВыберите способ формирования массива:\n");
    printf("1. Ввод с клавиатуры\n");
    printf("2. Генерация случайных чисел\n");
    printf("3. Чтение из файла\n");
    printf("Ваш выбор: ");
    scanf("%d", &choice);
    return choice;
}

// Ввод массива с клавиатуры
int* input_from_keyboard(int* size)
{
    printf("Введите размер массива: ");
    scanf("%d", size);
```

```
    int* arr = malloc(*size * sizeof(int));
    if (arr == NULL)
    {
        printf("Ошибка выделения памяти!\n");
        exit(1);
    }

    printf("Введите %d элементов массива:\n", *size);
    for (int i = 0; i < *size; i++)
    {
        printf("Элемент %d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    return arr;
}

// Генерация случайного массива
int* generate_random_array(int* size)
{
    printf("Введите размер массива: ");
    scanf("%d", size);

    int* arr = malloc(*size * sizeof(int));
    if (arr == NULL)
    {
        printf("Ошибка выделения памяти!\n");
        exit(1);
    }

    int min, max;
    printf("Введите минимальное значение: ");
    scanf("%d", &min);
    printf("Введите максимальное значение: ");
    scanf("%d", &max);

    srand(time(NULL));
    for (int i = 0; i < *size; i++)
    {
        arr[i] = rand() % (max - min + 1) + min;
    }

    return arr;
}

// Чтение массива из файла
int* read_from_file(int* size)
{
    char filename[100];
    printf("Введите имя файла: ");
    scanf("%s", filename);

    FILE* file = fopen(filename, "r");
    if (file == NULL)
    {
        printf("Ошибка открытия файла '%s'!\n", filename);
        exit(1);
    }

    int count = 0;
    int temp;

    while (fscanf(file, "%d", &temp) == 1)
    {
```

```
        count++;
    }

    if (count == 0)
    {
        printf("Файл не содержит чисел!\n");
        fclose(file);
        exit(1);
    }

    rewind(file);

    int* arr = malloc(count * sizeof(int));
    if (arr == NULL)
    {
        printf("Ошибка выделения памяти!\n");
        fclose(file);
        exit(1);
    }

    for (int i = 0; i < count; i++)
    {
        fscanf(file, "%d", &arr[i]);
    }

    fclose(file);
    *size = count;

    printf("Успешно прочитано %d элементов из файла '%s'\n", count, filename);
    return arr;
}


int main()
{
    setlocale(LC_ALL, "Russian");
    int* arr = NULL;
    int size = 0;
    int sort_method, input_method;

    printf("Программа для сортировки массивов\n");

    // Выбор метода ввода данных
    input_method = select_input_method();

    switch (input_method)
    {
    case 1:
        arr = input_from_keyboard(&size);
        break;
    case 2:
        arr = generate_random_array(&size);
        break;
    case 3:
        arr = read_from_file(&size);
        break;
    default:
        printf("Неверный выбор!\n");
        return 1;
    }

    printf("\nИсходный массив (%d элементов):\n", size);
    print_array(arr, size);

    // Выбор метода сортировки
```

```
    sort_method = select_sort_method();

    printf("\nСортировка...\n");

    // Замер времени выполнения
    clock_t start_time = clock();

    switch (sort_method)
    {
    case 1:
        quicksort(arr, 0, size - 1);
        printf("Применена быстрая сортировка\n");
        break;
    case 2:
        mergeSort(arr, size);
        printf("Применена сортировка слиянием\n");
        break;
    default:
        printf("Неверный выбор!\n");
        free(arr);
        return 1;
    }

    clock_t end_time = clock();
    double time_taken = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;

    printf("\nОтсортированный массив:\n");
    print_array(arr, size);

    printf("\nВремя выполнения: %.6f секунд\n", time_taken);

    free(arr);

    return 0;
}
```

Листинг 1. Код программы

Тестирование:

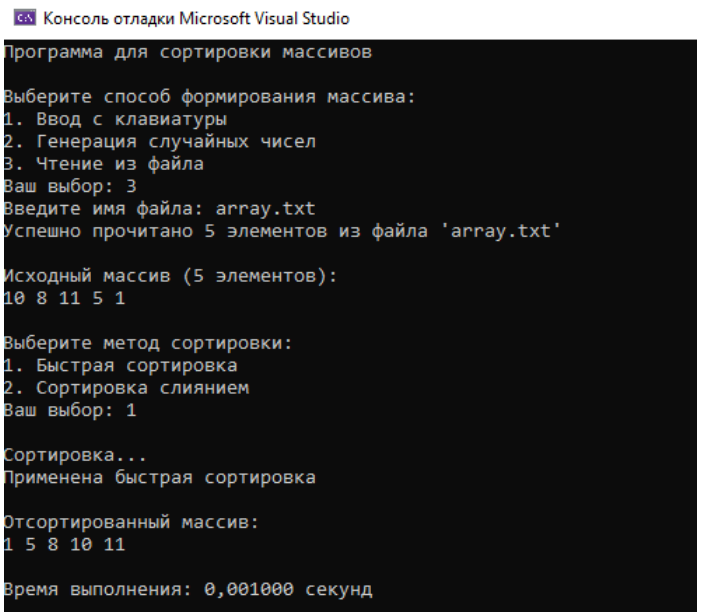

Рисунок 1. Результат быстрой сортировки при чтении массива из файла.

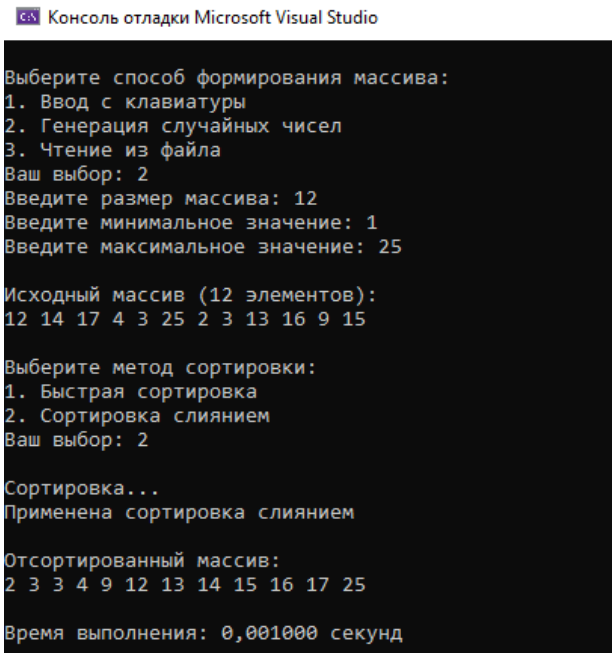

Рисунок 2. Результат сортировки слиянием при генерации случайного массива.