Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Сортировки»**

**Выполнил**:

студент группы 3821Б1ПМ2

Фомичев Дмитрий Евгеньевич

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**СОДЕРЖАНИЕ**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# ПОСТАНОВКА ЗАДАЧИ

― Для типа данных float реализовать сортировки: bubble sort, quick sort, merge sort, radix sort. Проверить сортировки на корректность с помощью готовой функции сортировки в синтаксисе языка Си.

― Замерить число перестановок и количество сравнений при сортировках.

― Провести эксперименты, чтобы показать теоритическую сложность алгоритмов.

― Написать вывод о проведенной работе.

# МЕТОД РЕШЕНИЯ

BUBBLE SORT

Алгоритм пузырьковой сортировки состоит из повторяющихся проходов по массиву, в каждый из которых сравниваются два соседних элемента. Если они стоят не правильно, то меняем их местами. После **N-1** проходов, где **N** – количество элементов в массиве, элементы становятся на свои места, включая N-ый элемент. Установка флага позволит избежать лишних проходов по массиву, если он уже отсортирован.

QUICK SORT

Алгоритм быстрой сортировки Хоара построен на идее «разделяй и властвуй». Из массива выбирается некоторый элемент, который принимается в качестве опорного элемента (зачастую это средний элемент). Элементы, меньше опорного, располагаются до него, больше - после. После выполнить рекурсивно ту же последовательность действий для подмассивов.

MERGE SORT

Алгоритм сортировки слияния основан на идее объединения двух отсортированных массивов. Массив рекурсивно разбивается на меньшие массивы до тех пор, пока длина массива не достигнет единицы. Два самых маленьких массива сортируются и сливаются в один. Так происходит, пока исходный массив не будет отсортирован.

RADIX SORT

Алгоритм поразрядной сортировки основан на идее разбиения элементов массива на k-значное число, каждая цифра которого находится в диапазоне от 0 до m-1. Сортировка производится поразрядно с помощью алгоритмов устойчивых сортировок (например, сортировка подсчетом).

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для BUBBLESORT.sln

Запустите программу. На экране появится сгенерированный массив, ниже отсортированный. В самом низу появится строчка, показывающая количество сравнений (рис. 1).

Для QUICKSORT.sln, MERGESORT.sln, RADIXSORT.sln

Запустите программу. На экране появится меню выбора (рис. 2). Выберете необходимый массив, нажав на цифры, указанные в меню. После этого на экране появятся сгенерированный массив, ниже отсортированный. В самом низу появятся строчки, показывающие количество сравнений и перестановок (рис. 3). Для RADIXSORT только количество перестановок (рис. 4).

Рисунки в приложении

# ОПИСАНИЕ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

BUBBLESORT.cpp

Содержит одну функцию *main()*. В ней содержится генератор массива, сортировка, вывод отсортированного массива и подсчет количества сравнений.

QUICKSORT.CPP

*Main()* содержит меню выбора типа генерации массива, вызов функции *qs(a)*, вывод отсортированного массива и подсчет количества сравнений и перестановок. Функция *qs(float\* array)* принимает на вход сгенерированный массив *array* и возвращает уже отсортированный массив *a* в *main()*. Переменные *first* и *last* определяют номер элемента массива *array*. Массив и переменные подставляются в функцию *quicksort(array, first, last)*. Эта функция служит упрощением ввода для вызова функции *quicksort()*. Функция *quicksort(float\* array, int first, int last)* является быстрой сортировкой Хоара. На вход получает сгенерированный массив *array*, первый и последний элементы массива. Возвращает отсортированный массив. Функция *swap(float\* array, int left, int right)* на вход принимает массив и два соседних элемента массива. Функция обменивает значения двух элементов и возвращает их. Функция *generate\_array(float\* array)* создает случайный массив. На вход – «пустой» массив, на выход - заполненный случайными значениями.

MERGESORT.c

Функция *main()* содержит меню выбора типа генерации массива, вызов функции *ms(a)*, вывод отсортированного массива, сравнение элементов массива, отсортированных моим алгоритмом, и элементов массива, отсортированных с помощью готовой функции *qsort()*, и подсчет количества сравнений и перестановок. Функция compared(const void\* a, const void\* b) нужна для qsort(), чтобы сравнивать два соседних элемента. На вход два числа, на выход 1, -1 или 0. Функция *ms(float\* array)* принимает на вход сгенерированный массив *array* и возвращает уже отсортированный массив *a* в *main()*. Переменные *first* и *last* определяют номер элемента массива *array*. Массив и переменные подставляются в функцию *mergesort(array, first, last)*. Эта функция служит упрощением ввода для вызова функции *mergesort()*. Функция *mergesort(float\* array, int first, int last)* является сортировкой слиянием. Функция рекурсивно вызывает себя, а также функцию *merge*(). Функция *merge()* – слияние двух подмассивов. На вход получает массив и номера первого и последнего элементов. Возвращает массив. Функция *swap()* ни на что не влияет(забыл удалить из конечного кода). Функция *sort\_array(float\* array)* создает обратно отсортированный массив из чисел от 1 до 100. На вход - массив, на выход – массив. Функция *generate\_array(float\* array)* создает случайный массив. На вход – «пустой» массив, на выход - заполненный случайными значениями.

RADIXSORT.c

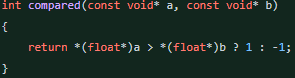
Функция *main()* содержит меню выбора типа генерации массива, вызов функции *rs(a)*, вывод отсортированного массива, сравнение элементов массива, отсортированных моим алгоритмом, и элементов массива, отсортированных с помощью готовой функции *qsort()*, и подсчет количества сравнений и перестановок. Функция compared(const void\* a, const void\* b) нужна для qsort(), чтобы сравнивать два соседних элемента. На вход два числа, на выход 1, -1 или 0. Функция *rs(float\* array)* принимает на вход сгенерированный массив *array* и возвращает уже отсортированный массив *a* в *main()*. Входной массив, а также дополнительный массив и число элементов в исходном массиве подставляются в функцию *radixSort().* Эта функция служит упрощением ввода для вызова функции *radixSort()*. Функция *radixSort(float\* in, float\* out, int N)* принимает на вход сгенерированный массив, дополнительный массив и число элементов в этих массивах. На выход подается отсортированный массив. Данная функция является поразрядной сортировкой. Функция *createCounters(float\* data, long\* counters, long N)* принимает на вход два массива, исходный и пустой, и количество элементов в них. Возвращаем оба массива. Функция является сортировкой подсчетом для всех разрядов. Функция *radixPass(short int Offset, long int N, float\* source, float\* dest, long\* count)* сортирует по одному разряду. На вход два массива, количество элементов в них и *Offset* – номер сортируемого разряда. Функция *signedradixLastPass(short Offset, long N, float\* source, float\* dest, long\* count)* позволяет сортировать знаковые числа. На вход два массива и массив из *createCounters(),* а также сортируемый разряд и число элементов в массивах.Функция *sort\_array(float\* array)* создает обратно отсортированный массив из чисел от 1 до 100. На вход - массив, на выход – массив. Функция *generate\_array(float\* array)* создает случайный массив. На вход – «пустой» массив, на выход - заполненный случайными значениями.

# ПОДТВЕРЖДЕНИЕ КОРРЕКТНОСТИ

Для подтверждения корректности в программе была написана сортировка из библиотеки stdlib.h языка Си – *qsort()*. В случае неверной сортировки мной написанной программы на экран выведется сообщение о некорректности данной сортировки, а также сам(и) элемент(ы), отличающийся(-еся) от элементов сортировки из библиотеки. Это обеспечивает алгоритм, который сравнивает каждый элемент отсортированного моей сортировкой массива с элементом сортировки *qsort()* (рис. 5). После нескольких тестов написанных сортировок ни одна не вызвала сообщения о некорректности. Отсюда я делаю вывод, что сортировки исправно работают.



Функция qsort()



Функция, отвечающая за сравнение в qsort(). В случае если а > b, то возвращает 1, a == b – 0, a < b – (-1).

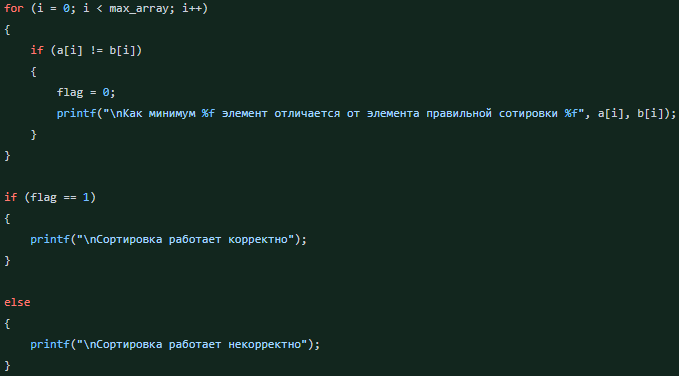


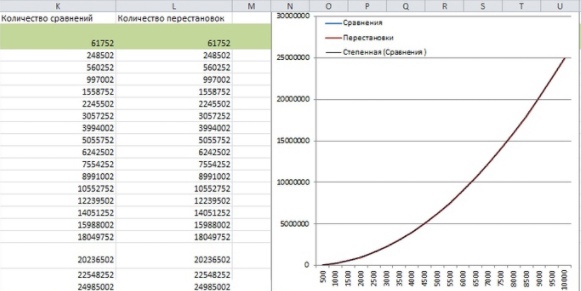
Рис. 5

# РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

BUBBLE SORT

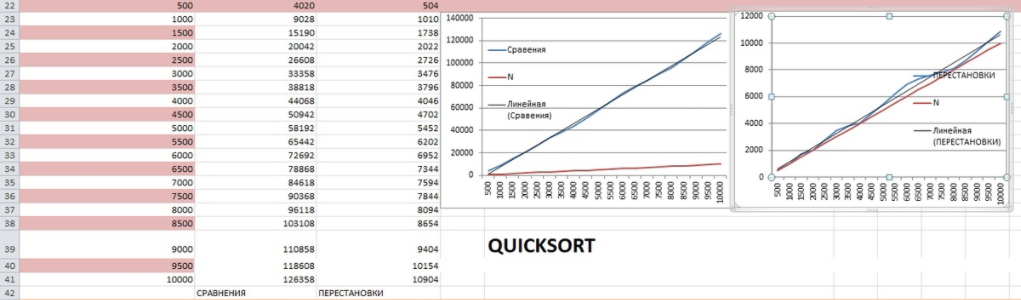


По данным эксперимента, для худшего случая (массив обратно отсортирован) количество и сравнений и перестановок увеличивалось в N^2 (для оценки я использовал степенную линию тренда в Excel) каждые 500 элементов. Это подтверждает сложность О(N^2) для худшего случая.



Для массива, где первая половина элементов расположены в порядке возрастания, а вторая половина в обратном, также каждые 500 элементов количество сравнений и перестановок увеличивалось в N^2 раз (для оценки я использовал степенную линию тренда в Excel). Это подтверждает сложность пузырьковой сортировки О(N^2) в среднем.

QUICK SORT

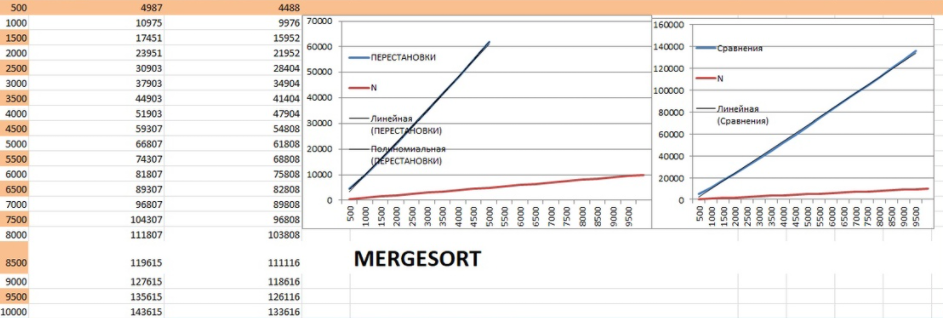


Для быстрой сортировки средний случай выполнялся за 0.89\*log(N)\*N сравнений и N перестановок при шаге в 500 элементов (от 500 до 10тыс.). Это соответствует сложности О(n\*log(n)).



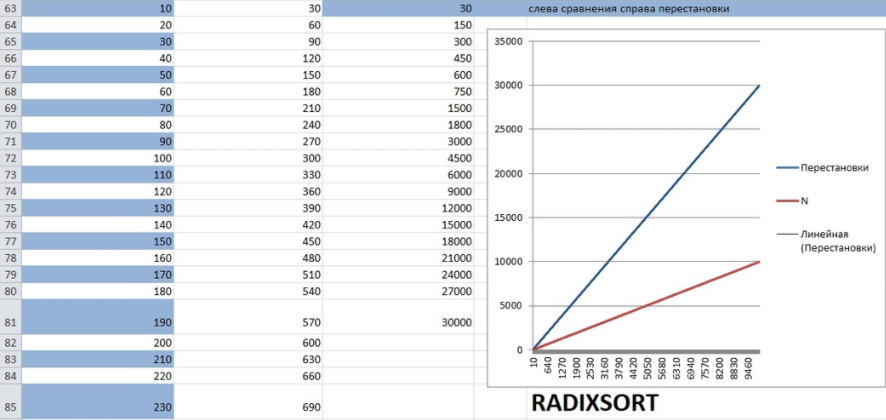
Один из наихудших вариантов для быстрой сортировки, когда опорный элемент это самый первый, а массив уже отсортирован. Для сравнений получилось N^2 (степенная линия тренда полностью копирует функцию). Для перестановок N. Это соответствует сложности О(N^2).

MERGE SORT



Количество сравнений 1,11\*N\*log(N). Количество перестановок N\*log(N). Это подтверждается графиком, который соответствует графику n\*log(n). Поэтому сложность для среднего случая О(n\*log(n)).

RADIX SORT



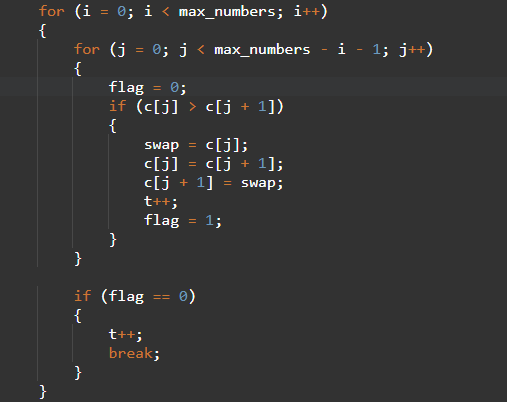
В сортировке нет сравнений. Перестановки увеличивались в 3 \* N при шаге 10 элементов(от 10 до 10тыс. элементов). Это соответствует сложности О(N).

# ЗАКЛЮЧЕНИЕ

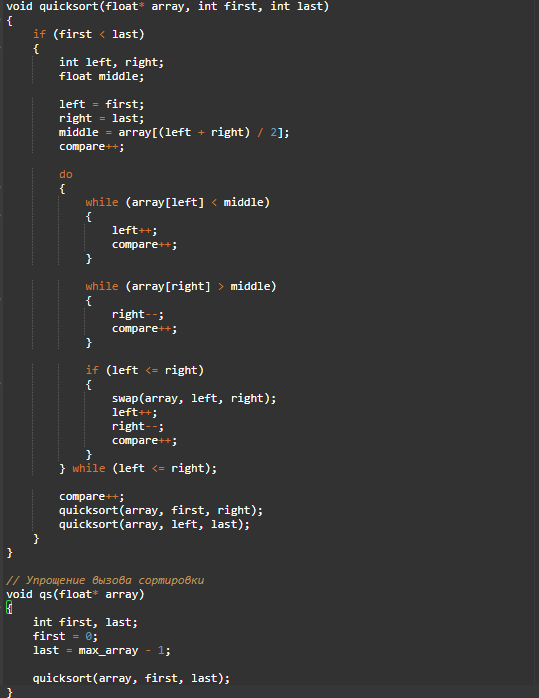
Я реализовал и исследовал 4 вида сортировок для типа данных float: сортировка пузырьком, быстрая сортировка Хоара, сортировка слиянием и поразрядная сортировка. Проверил всех их на корректность работы с помощью встроенного в библиотеки языка Си алгоритма сортировки. Подсчитал количество перестановок и сравнений в алгоритмах для разного количества элементов в сортируемом массиве. Также экспериментально подтвердил теоретическую сложность всех алгоритмов с помощью таблицы и графиков.

# ПРИЛОЖЕНИЕ

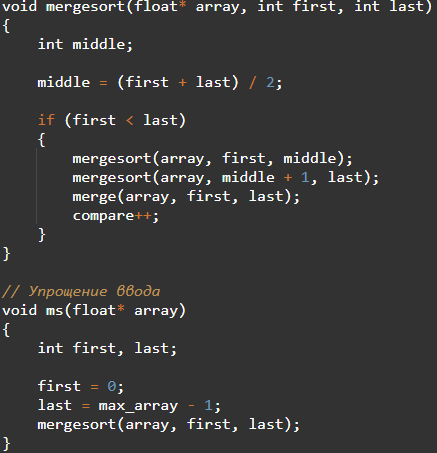
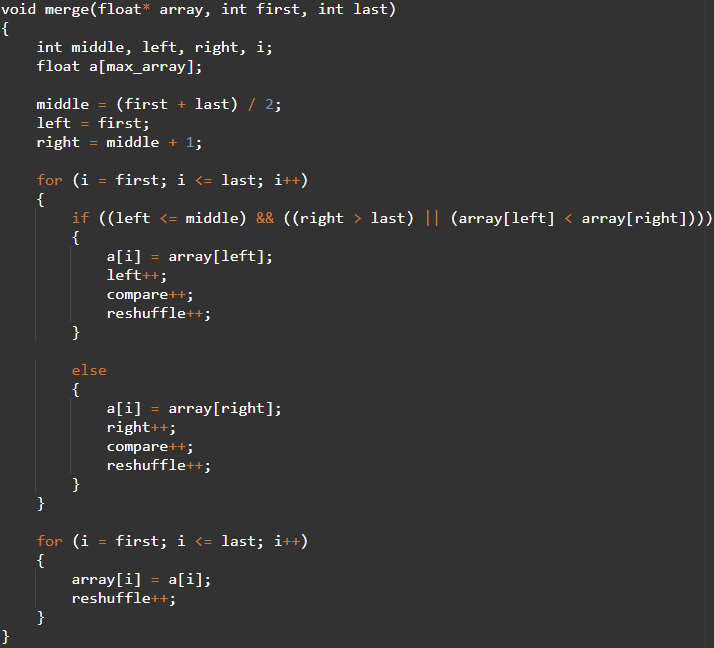
BUBBLE SORT



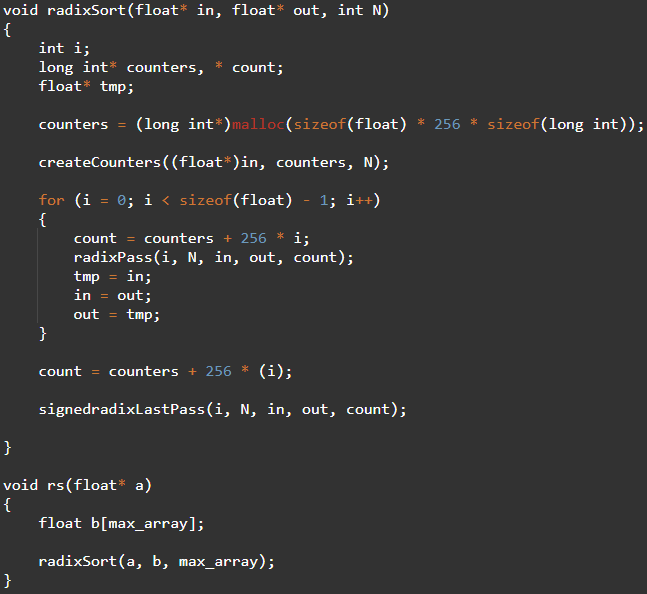
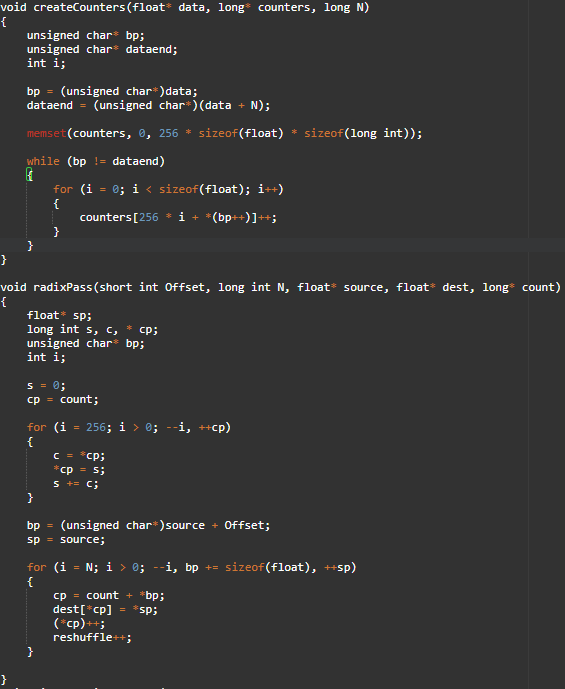
QUICK SORT

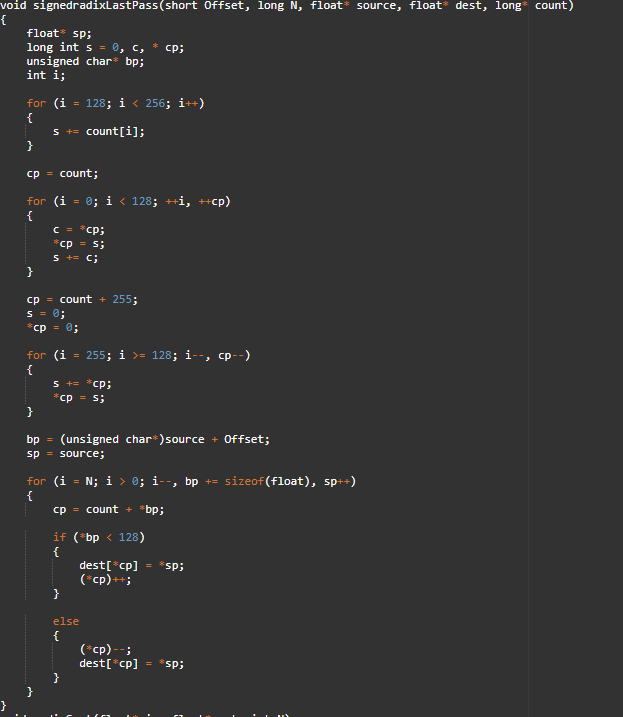


MERGE SORT

RADIX SORT



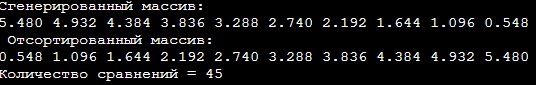


Рис. 1

https://sun9-27.userapi.com/impg/MdcL4HC7Og1Uvi7947kb1FcL-w-Q1CS9WGnBFQ/CFFCzSAucI8.jpg?size=610x28&quality=96&sign=421d0ebf39994d1d9596203281843b5a&type=album

Рис. 2

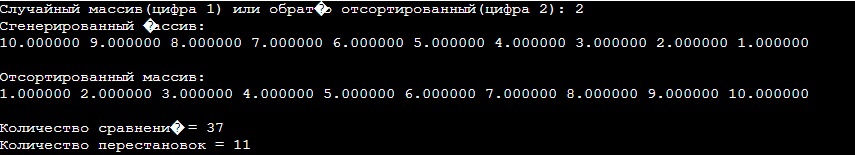


Рис. 3

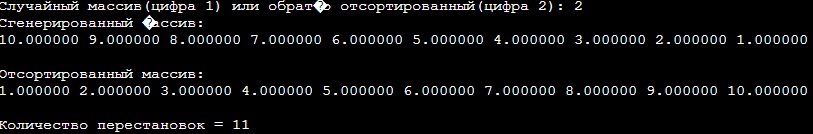


Рис. 4