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 10](#_Toc26962566)

[Результаты экспериментов 11](#_Toc26962567)

[Заключение 17](#_Toc26962568)

[Приложение 18](#_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

ALLSORTS.sln

Запустите программу. На экране появится меню выбора типа генерации массива (рис. 1). Выберите необходимый массив, нажав на цифры, указанные в меню. После этого на экране появятся сообщение, о выбранном массиве. Чтобы выйти из программы, нажмите 0. Перед следующим меню появится информация, о выбранном массиве на предыдущем шаге. В новом меню вам необходимо выбрать сортировку (рис. 2). Появится сообщение о выбранной сортировке. Чтобы перейти на предыдущий шаг, нажмите 0. Перед выбором количества элементов в массиве появится информация о выбранных на предыдущих шагах настройках. Выберите количество элементов (рис. 3). На экране появится сгенерированный массив, отсортированный массив, сообщение о корректности сортировки, количество сравнений и перестановок (рис. 4). Чтобы перейти на предыдущий шаг, нажмите 0.

Рисунки в приложении

# ОПИСАНИЕ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

ALLSORTS.cpp

*main()*

Функция содержит функцию установки русского языка *setlocale(LC\_ALL, "Russian")* и вызов функции *menu()*.

*menu()*

Функция содержит реализацию меню выбора: генерации массива, сортировок и количества элемента в массиве. Все реализовано с помощью цикла while и оператора выбора switch/case. Содержит вызов функций *inform()* и *sort\_menu()*.

*inform(short int alg\_select, short int sort\_select, int max\_array)*

Функция, основная задача которого вывод информации о текущем выборе генерации массива и вида сортировки. Принимает значения переменных alg\_select, *sort\_select* и *max\_array*, которые были выбраны в *menu()*, и выводит на экран соответствующие информацию.

*sort\_menu(short int alg\_select, short int sort\_select, int max\_array)*

Функция, исполняющая основную работу. Создает два массива с помощью функции *calloc()*, количество элементов в которых определяются с помощью переменной *max\_array.* Наполняет первый массив с помощью выбранной генерации(*alg\_select* определяет тип генерации). Дублирование элементов во второй массив. С помощью  *sort\_select* определяется сортировка. Выводятся элементы первого массива после сортировки. С помощью функции *correct()* отсортированный массив проверяется на корректность. На экран выводится информация по количеству сравнений и перестановок в сортировках. Переменные *compare* и *reshuffle* получают значение 0, созданные массивы удаляются с помощью функции *free()*.

*correct(float\* a, float\* b)*

Функция проверяет корректность сортированного массива. Информация о ней в разделе **ПОДТВЕРЖДЕНИЕ КОРРЕКТНОСТИ.**

*sort\_array(float\* array)*

Функция создает обратно отсортированный массив из чисел от 1 до 100. На вход - массив, на выход – массив.

*generate\_array(float\* array)*

Функция создает случайный массив. На вход – «пустой» массив, на выход - заполненный случайными значениями.

*bs(float\* array)*

Функция является пузырьковой сортировкой. На вход сгенерированный массив, на выход отсортированный.

*qs(float\* array)*

Функция принимает на вход сгенерированный массив *array* и возвращает уже отсортированный массив *array* в *sort\_menu()*. 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()*.

*quicksort(float\* array, int first, int last)*

Функция является быстрой сортировкой Хоара. На вход получает сгенерированный массив *array*, первый и последний элементы массива. Возвращает отсортированный массив.

*swap(float\* array, int left, int right)*

Функция на вход принимает массив и два соседних элемента массива. Функция обменивает значения двух элементов и возвращает их.

*ms(float\* array)*

Функция принимает на вход сгенерированный массив *array* и возвращает уже отсортированный массив *array* в *sort\_menu()*. 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()*.

*mergesort(float\* array, int first, int last)*

Функция является сортировкой слиянием. Функция рекурсивно вызывает себя, а также функцию *merge*().

*merge()*

Функция отвечает за слияние двух подмассивов. На вход получает массив и номера первого и последнего элементов. Возвращает массив.

*rs(float\* array)*

Функция принимает на вход сгенерированный массив *array* и возвращает уже отсортированный массив *array* в *sort\_menu()*. 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()*.

*radixSort(float\* in, float\* out, int N)*

Функция принимает на вход сгенерированный массив, дополнительный массив и число элементов в этих массивах. На выход подается отсортированный массив. Данная функция является поразрядной сортировкой.

*createCounters(float\* data, long\* counters, long N)*

Функцияпринимает на вход два массива, исходный и пустой, и количество элементов в них. Возвращаем оба массива. Функция является сортировкой подсчетом для всех разрядов.

*radixPass(short int Offset, long int N, float\* source, float\* dest, long\* count)*

Функциясортирует по одному разряду. На вход два массива, количество элементов в них и *Offset* – номер сортируемого разряда.

*signedradixLastPass(short Offset, long N, float\* source, float\* dest, long\* count)*

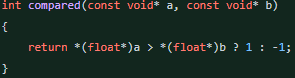
Функция позволяет сортировать знаковые числа. На вход два массива и массив из *createCounters(),* а также сортируемый разряд и число элементов в массивах.

# 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).

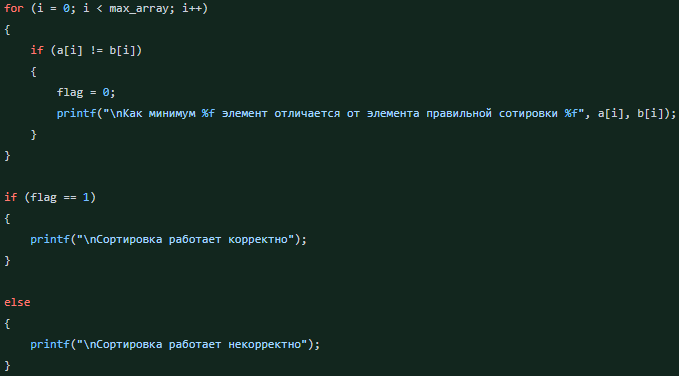
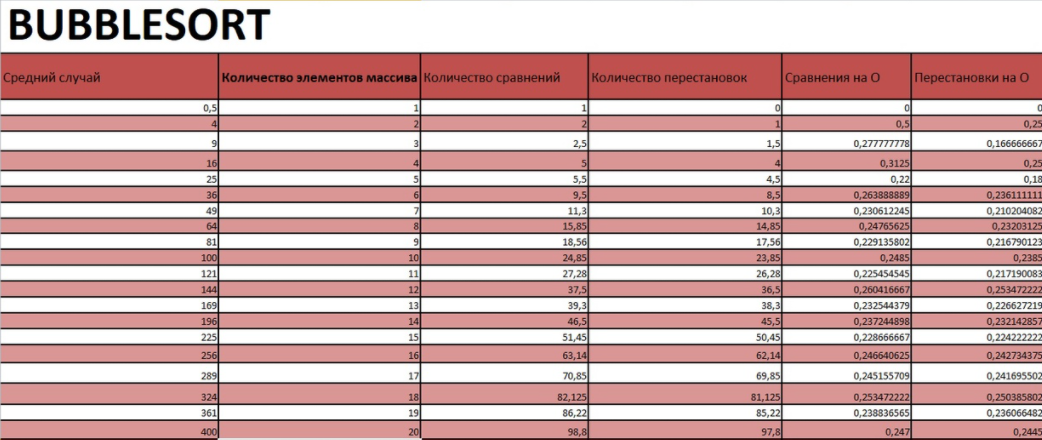


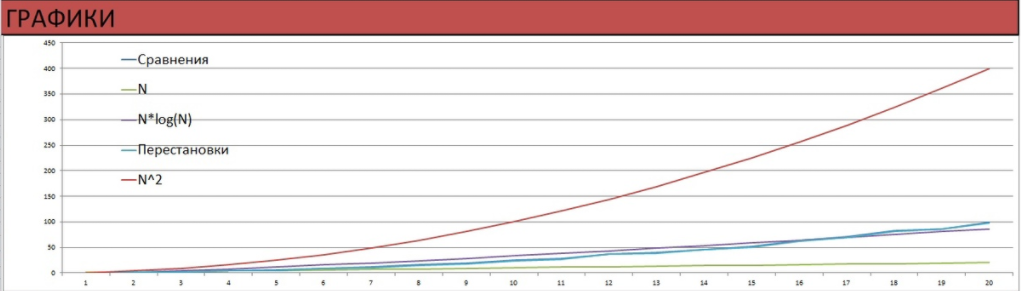
Рис. 5

# РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

**BUBBLE SORT**

Средний случай



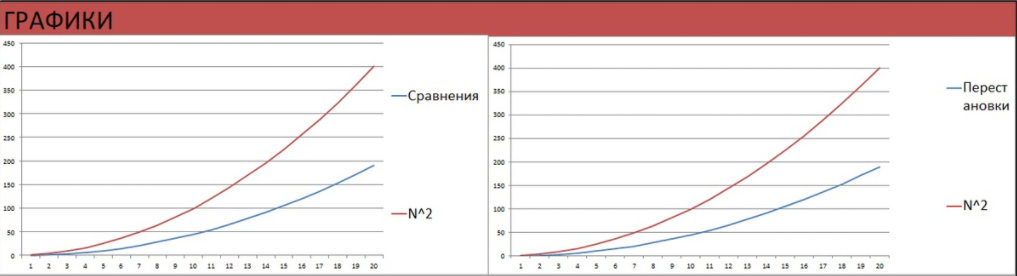


Для пузырьковой сортировки были высчитаны средние значения сравнений и перестановок при сортировке случайно сгенерированного массива. Количество сравнений больше количества перестановок на 1. Функция сравнений приблизительно равнялась функции N\*log(N), где N – количество элементов массива. В таблице приведены значения отношений количества перестановок и сравнений к функции N^2.

Вывод: теоретическая сложность сортировки пузырьком при случайных данных равна O(N\*log(N)).

**Худший случай**



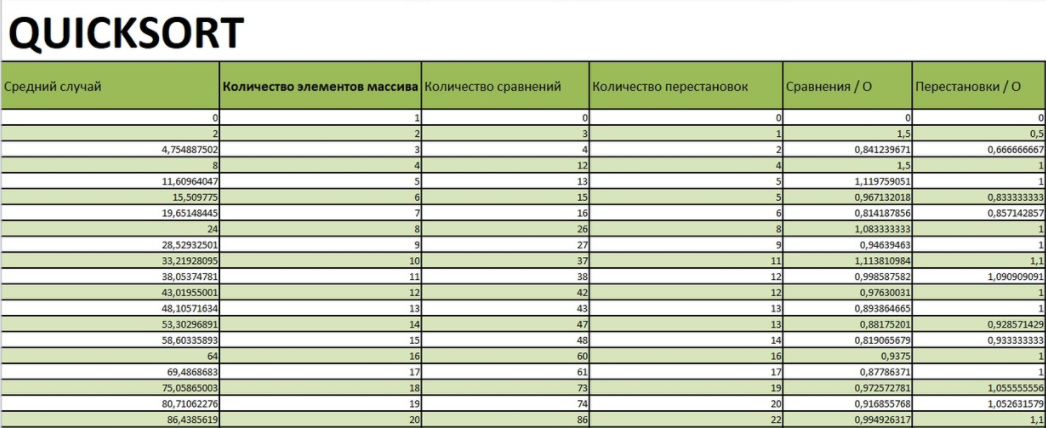


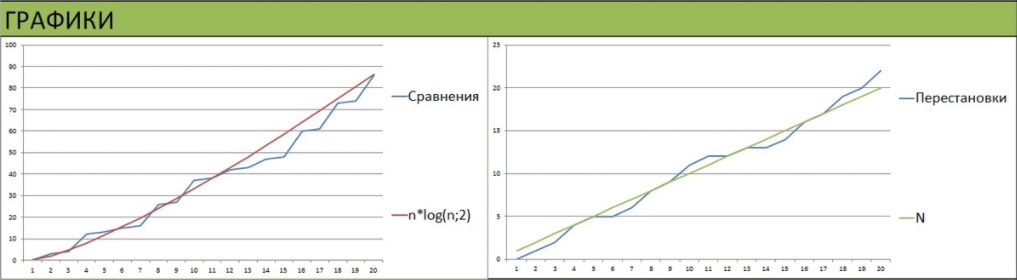
Худший случай для быстрой сортировки - обратно отсортированный массив. Количество сравнений и перестановок одинаково. Отношения количества сравнений (или перестановок) к N^2 начинаются от 0,25 и стремятся к 0,5.

Вывод: худший случай сортировки пузырьком действительно имеет теоретическую сложность О(N^2).

**QUICKSORT**

**Средний случай**



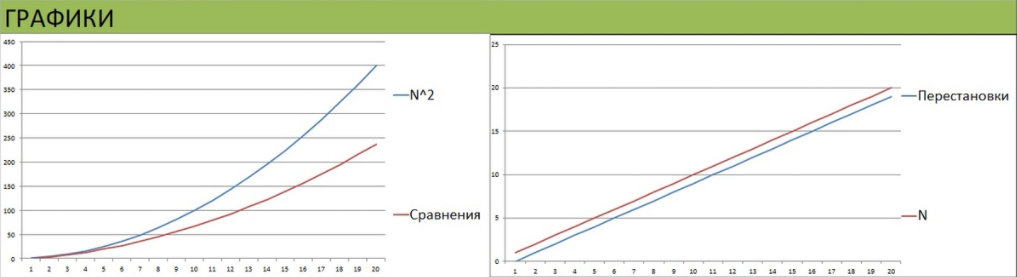


Быстрая сортировка тестировалась на обратно сортированном массиве. График функции Сравнения, где Сравнения – количество сравнений в сортировке, с небольшими отклонениями повторяет график функции N\*log(N), где N – количество элементов в массиве. Столбец Сравнения / О показывает эти отклонения. График функции Перестановки, где Перестановки – количество перестановок при сортировке, практически совпадает с графиком функции N. Отклонения приведены в таблице Перестановки / О.

Вывод: для среднего случая быстрой сортировки действительно теоретическая сложность равна О(N\*log(N)).

**Худший случай**





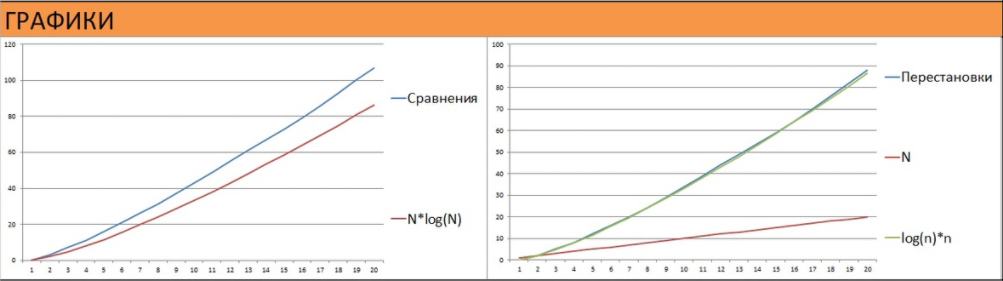
Худший случай для быстрой сортировки получается, если опорным элементом является самый последний, а сортируемый массив обратно отсортирован. Тогда количество сравнений принимают вид функции N^2, а перестановки вид функции N. Отношения количества сравнений к N^2 начинаются от 0,75... и стремятся к 0,5. Отношения количества перестановок к N начинаются от 0,5 и стремятся к 1.

Вывод: для худшего случая быстрой сортировки действительно теоретическая сложность равна O(N^2).

**MERGE SORT**

**Худший случай/Средний случай/Лучший случай**



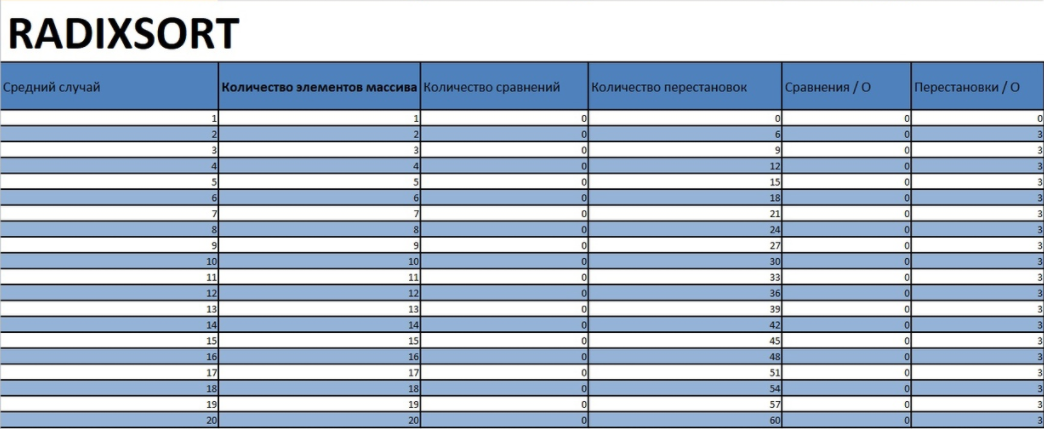


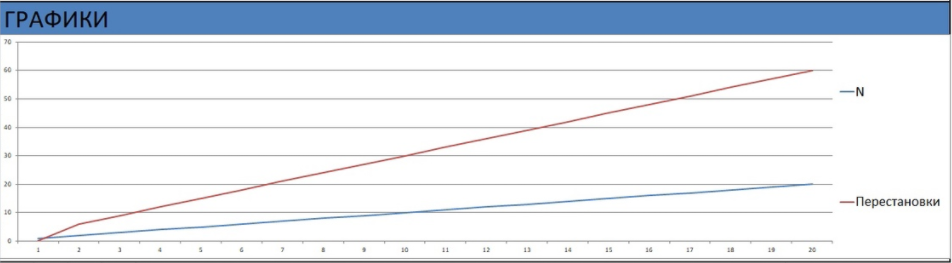
При любом стечении обстоятельств сортировка слиянием будет иметь сложность O(N\*log(N)). Сортировка тестировалась на обратно отсортированном массиве. Отношения количества сравнений к N\*log(N) начинаются от 1,5... и стремятся к 1,2. Отношения количества перестановок к N\*log(N) начинаются от 1,05... и стремятся к 1.

Вывод: сортировка слиянием действительно имеет теоретическую сложность О(N\*log(N)).

**RADIX SORT**

**Худший случай/Средний случай/Лучший случай**

****

****

Для худшего и среднего случая теоретическая сложность не меняется, но меняется объем дополнительной памяти. В сортировке нет сравнений, поэтому исследуются только перестановки. Так отношения количества перестановок к N принимают значение 3.

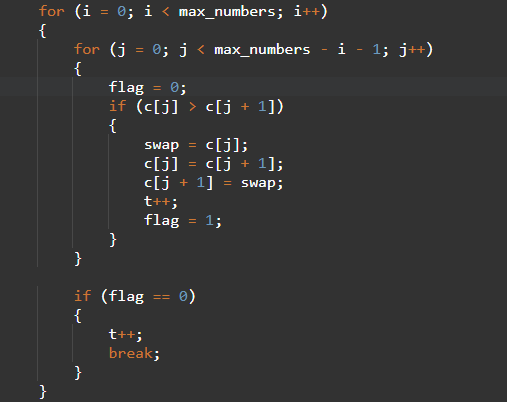
Вывод: для поразрядной сортировки теоретическая сложность действительно равняется О(N).

# ЗАКЛЮЧЕНИЕ

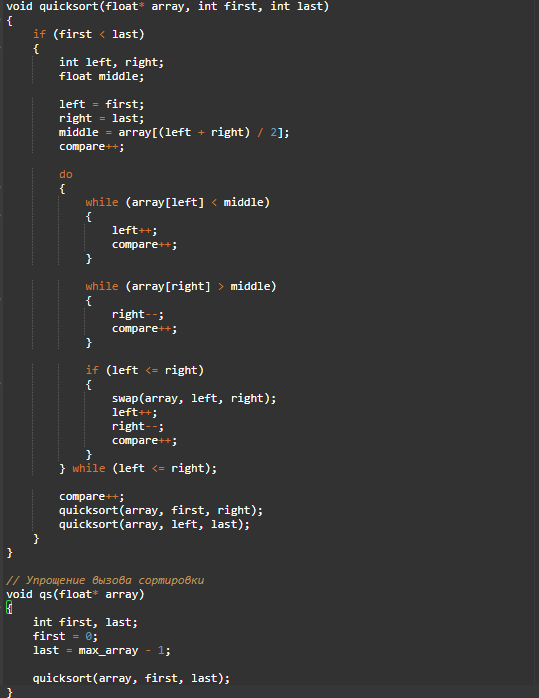
Я реализовал и исследовал 4 вида сортировок для типа данных float: сортировка пузырьком, быстрая сортировка Хоара, сортировка слиянием и поразрядная сортировка. Проверил всех их на корректность работы с помощью встроенного в библиотеке языка Си алгоритма сортировки. Подсчитал количество перестановок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

# ПРИЛОЖЕНИЕ

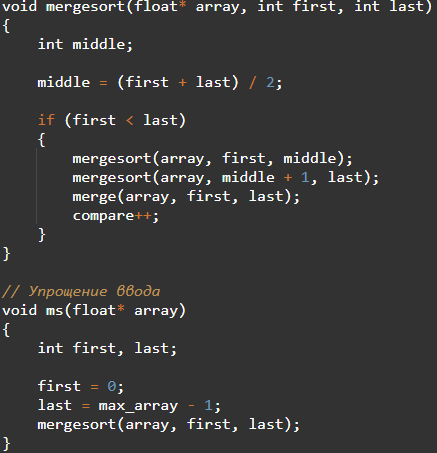
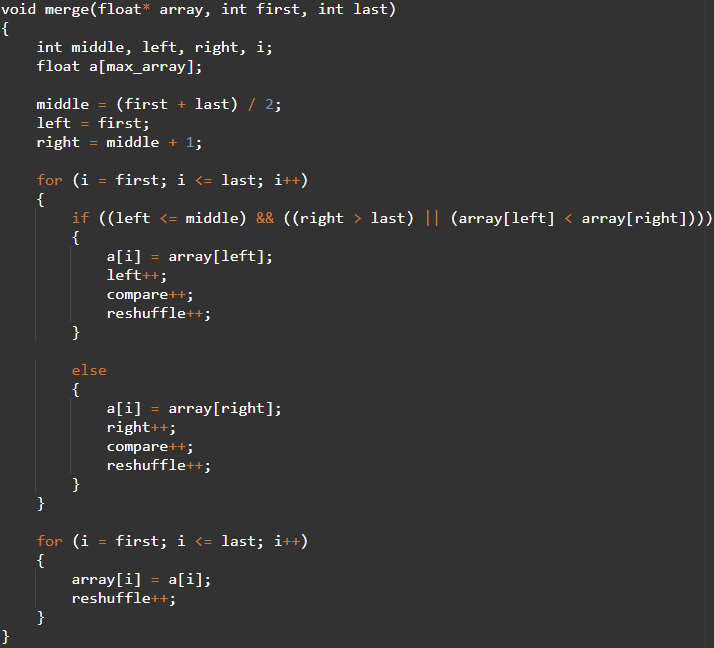
BUBBLE SORT



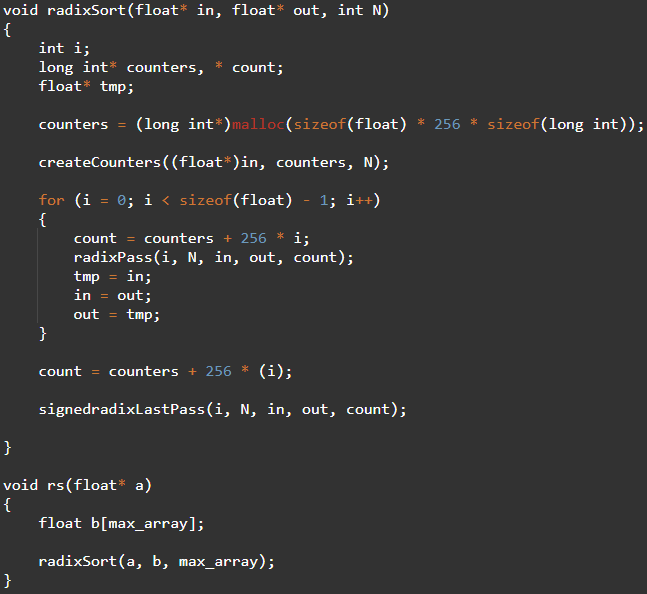
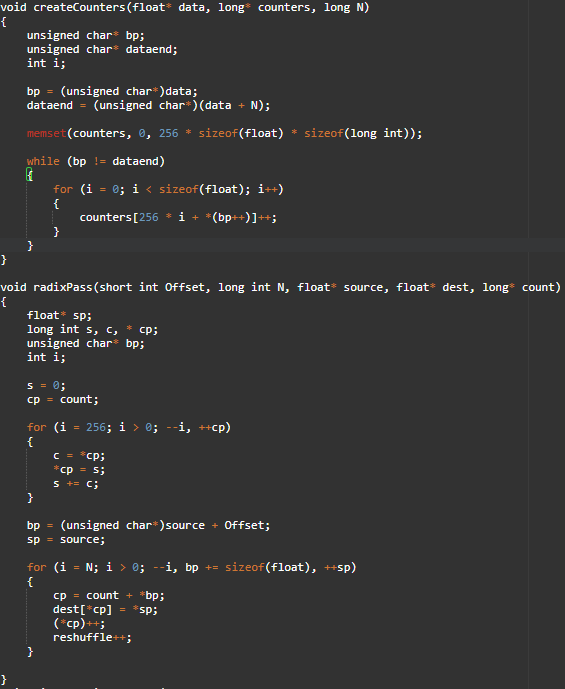
QUICK SORT

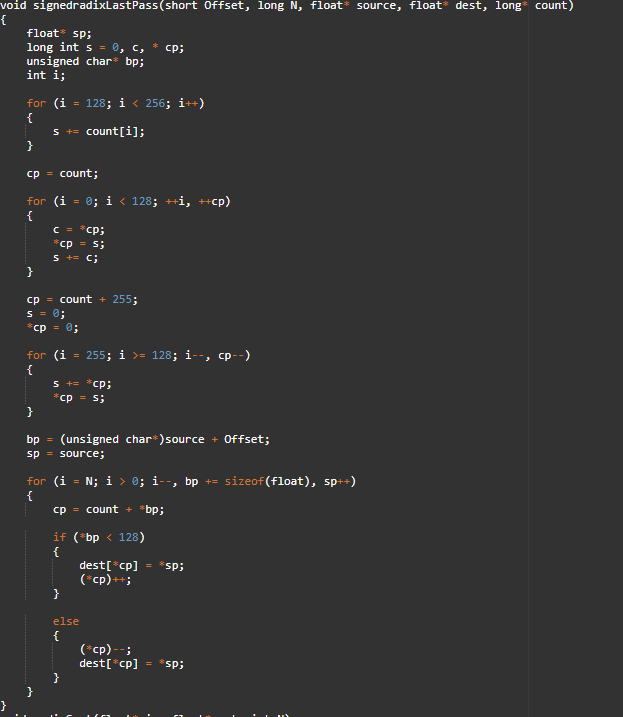


MERGE SORT

RADIX SORT



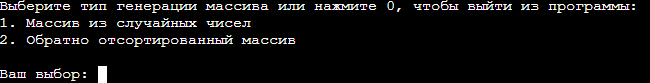


Рис. 1

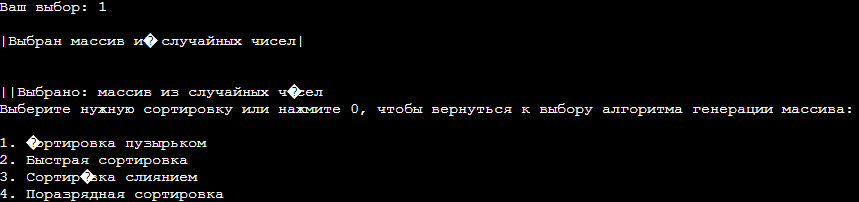


Рис. 2

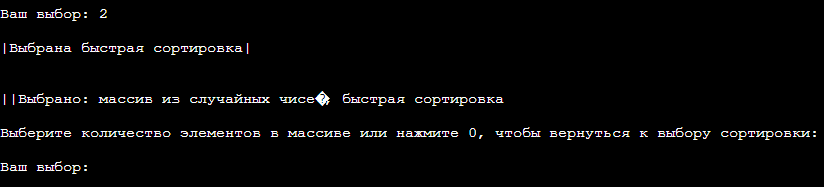


Рис. 3

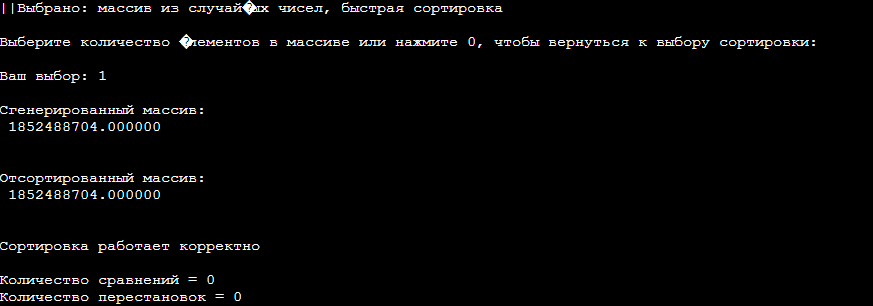


Рис. 4