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**

**«Алгоритмы сортировки массива»**

**Выполнил**:

студент группы 382003-1

Ягилев И.М

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020

**Содержание**

[Постановка задачи 3](file:///C:\Users\ИВан\Desktop\Otchet_Shablon.docx#_Toc26962562)

[Метод решения 4](file:///C:\Users\ИВан\Desktop\Otchet_Shablon.docx#_Toc26962563)

[Руководство пользователя 8](file:///C:\Users\ИВан\Desktop\Otchet_Shablon.docx#_Toc26962564)

[Описание программной реализации 10](file:///C:\Users\ИВан\Desktop\Otchet_Shablon.docx#_Toc26962565)

[Подтверждение корректности 11](file:///C:\Users\ИВан\Desktop\Otchet_Shablon.docx#_Toc26962566)

[Результаты экспериментов 12](file:///C:\Users\ИВан\Desktop\Otchet_Shablon.docx#_Toc26962567)

[Заключение 15](file:///C:\Users\ИВан\Desktop\Otchet_Shablon.docx#_Toc26962568)

[Приложение 16](file:///C:\Users\ИВан\Desktop\Otchet_Shablon.docx#_Toc26962569)

# Постановка задачи

Необходимо реализовать алгоритмы сортировки массивов для типа данных float. Замерить количество сравнений и перестановок при выполнении алгоритма. Поставить эксперименты, отражающие теоретическую сложность алгоритмов.

Сортировки, которые необходимо реализовать:

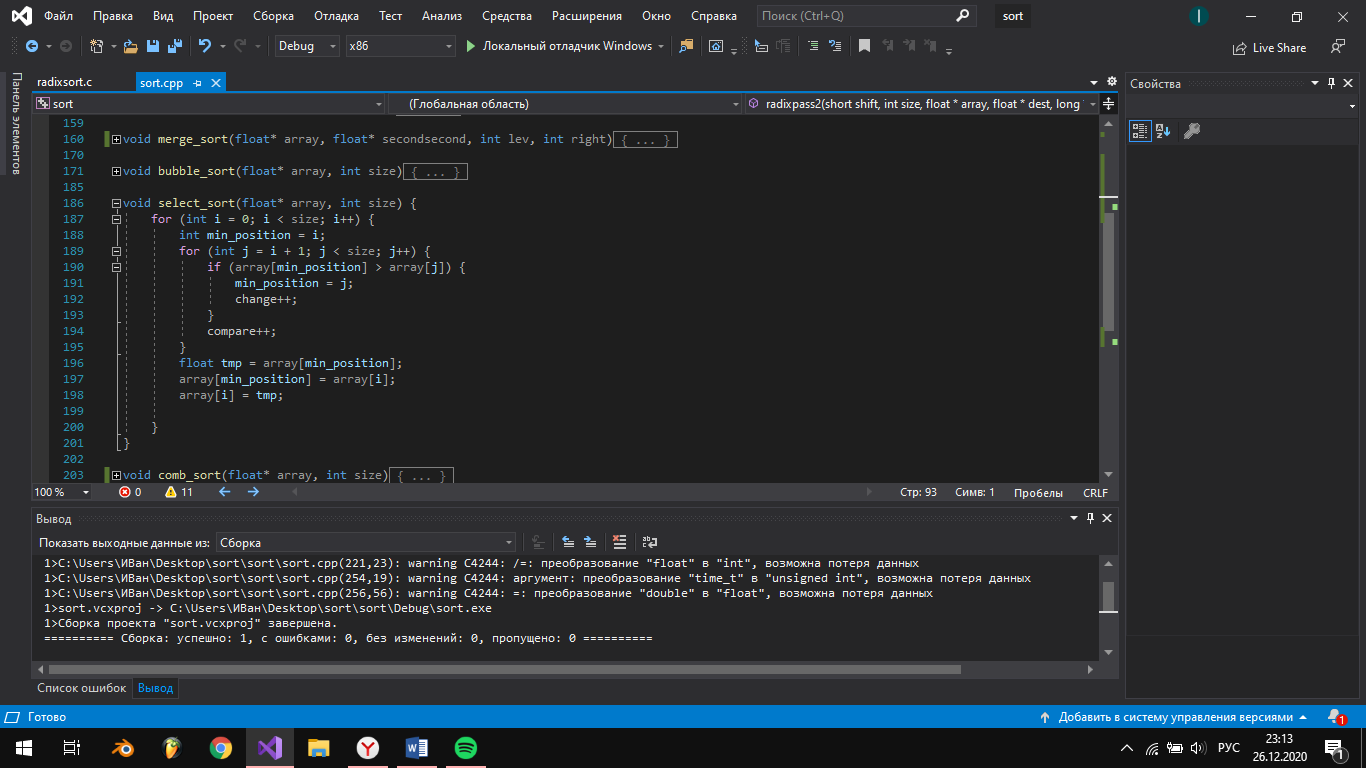
1. Сортировка выбором o (N^2);
2. Сортировка расчёской o (N\*log n);
3. Сортировка слиянием o (N\*log n);
4. Поразрядная сортировка o (N).

# Метод решения

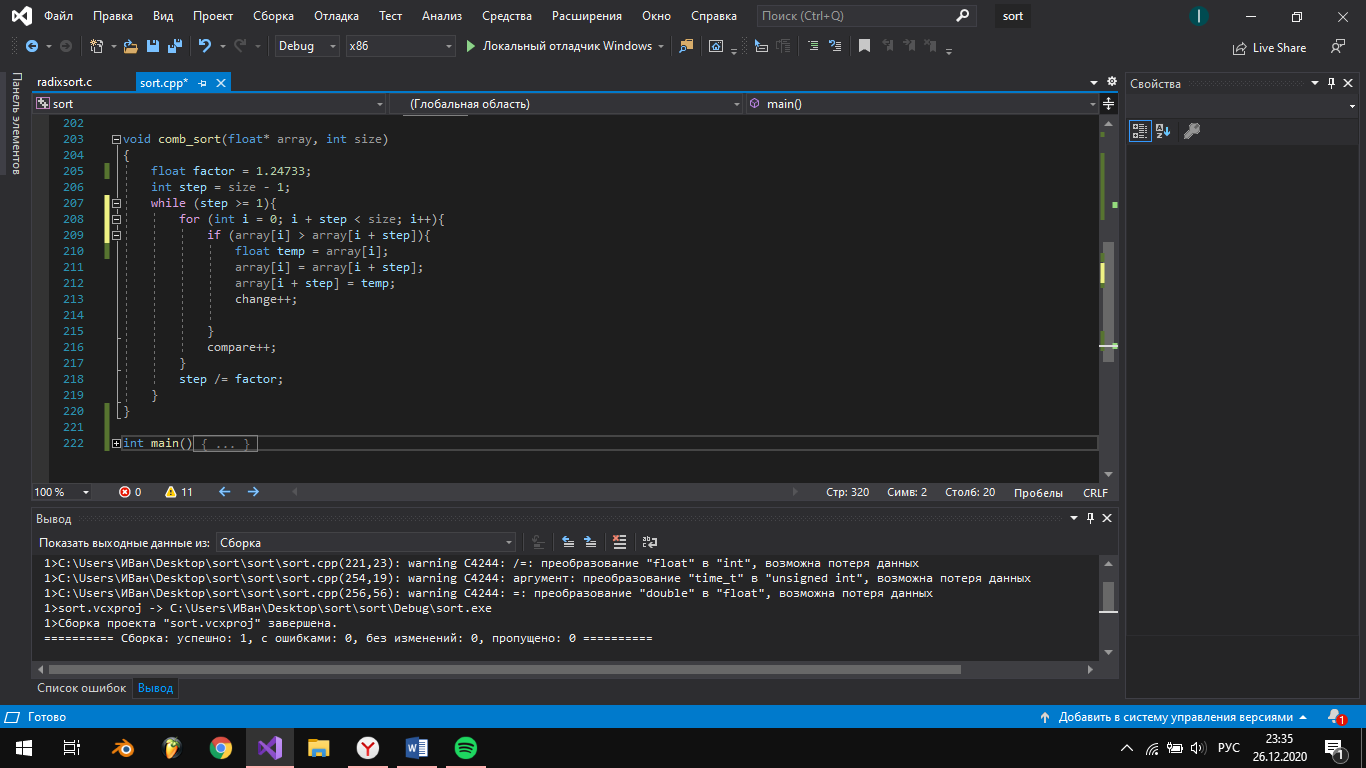
1. **Сортировка выбором**

Одна из простых сортировок, время выполнения в лучшем, худшем и среднем случаях равно o (N^2).

**Реализация:**

* ****Используем цикл for, проходящий по всем элементам массива. Первый элемент (i=0) –минимальная позиция;
* Используем второй цикл for, проходящий по всем элементам массива, начиная с i+1;
* Используем условие и сравниваем элементы друг с другом, тем самым находим индекс минимального элемента.
* После окончания работы второго цикла, меняем элементы массива местами, использую вспомогательную переменную.

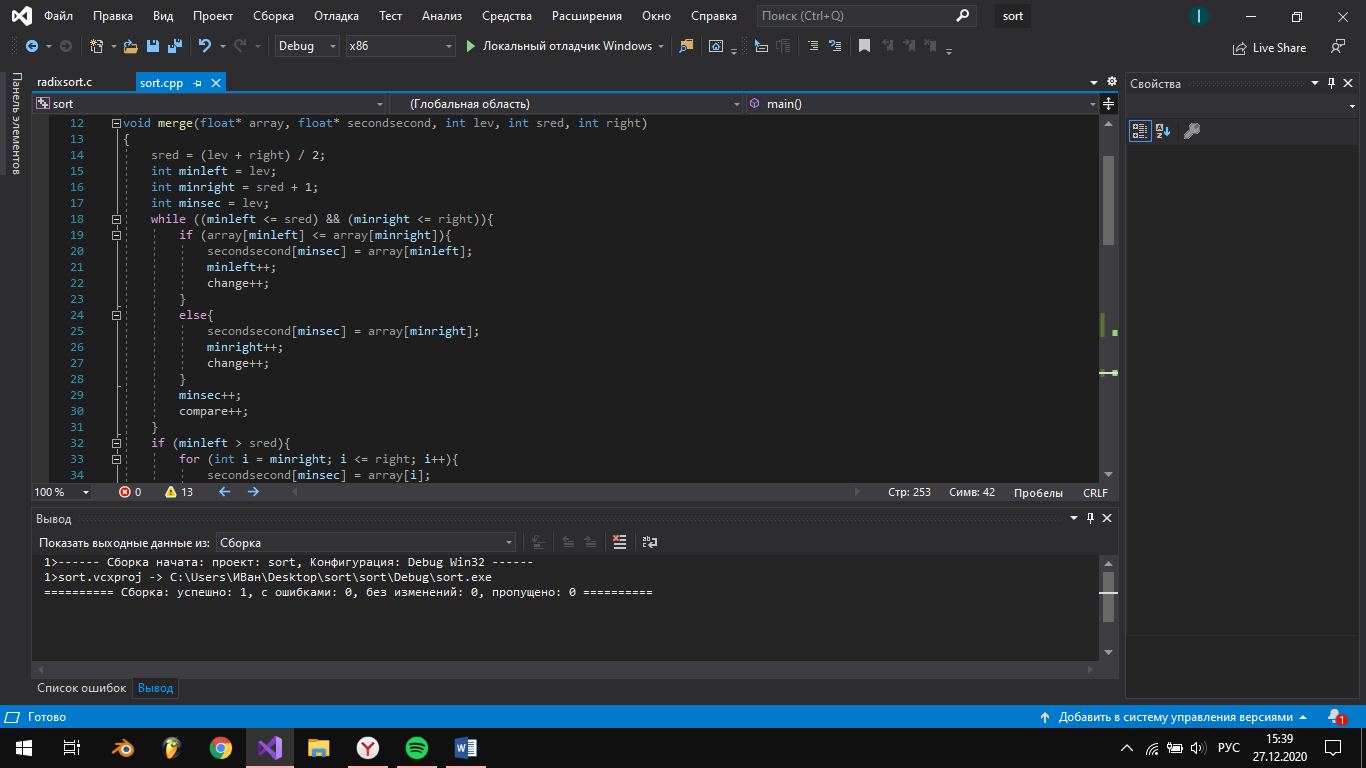
1. **Сортировка расчёской**

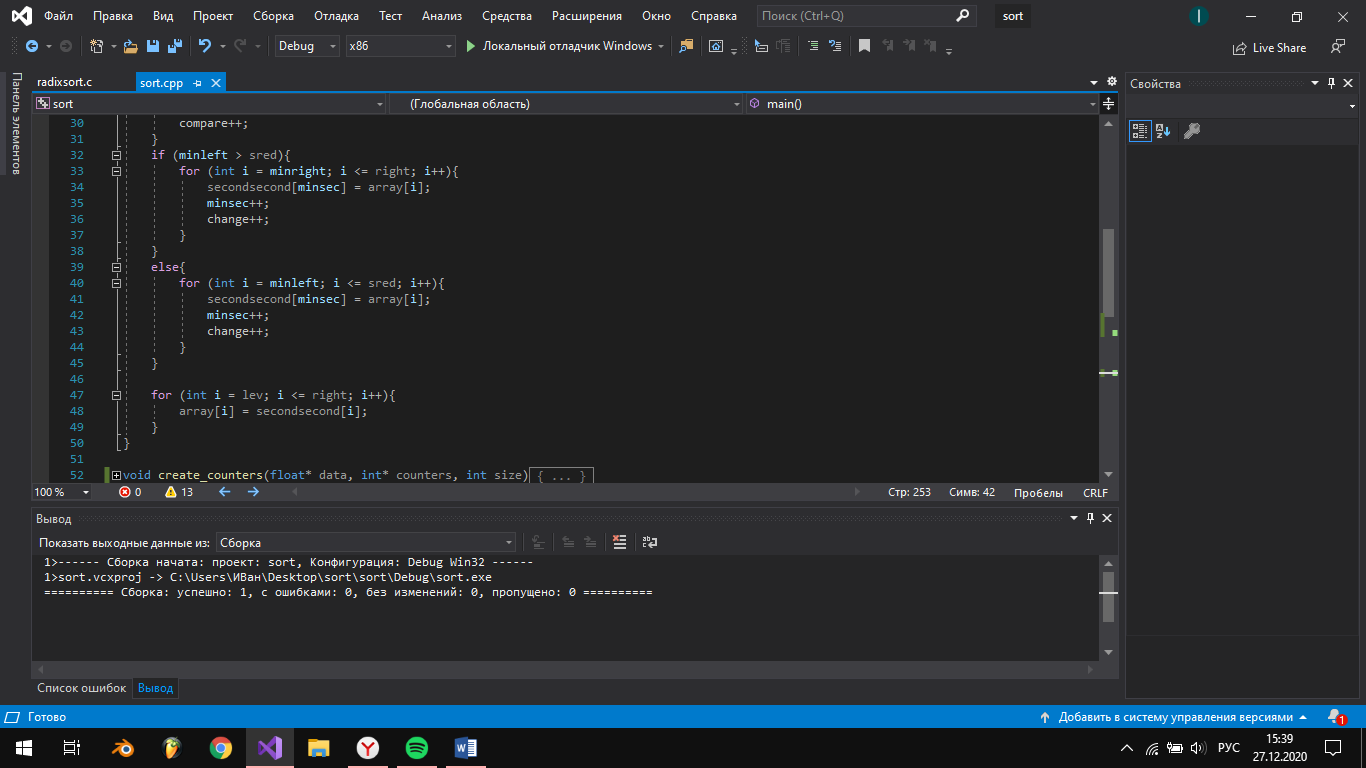
****Одна из быстрых сортировок, является модификации сортировки пузырьком. Основное отличие – первоначально брать достаточно большое расстояние между сравниваемыми элементами и по мере упорядочивания массива сужать это расстояние вплоть до минимального. Первоначально расстояние между сравниваемыми элементами максимально (размер массива - 1), затем, деля расстояние на специальную величину, мы получаем новый шаг.

**Реализация:**

* Как сказано выше, инициализируется специальная величина (1.24733…), выбирается максимальный шаг (size - 1);
* Запускаем цикл while, который контролирует, чтобы шаг был не меньше единицы, то есть, чтобы сортировка в конечном счете стала сортировкой пузырьком;
* Входим в цикл for, проходящий по всем элементам массива, дополнительно ставим ограничение, чтобы сумма индекса и шага была меньше размера массива;
* Условие, сравнивающее два выбранных элемента массива, в случае выполнения условия, элементы меняются местами;
* Получение нового шага, путем деления на специальную величину.

1. **Сортировка слиянием**

Одна из быстрых сортировок, принцип работы достаточно прост – задача сортировки подразделяется на несколько мелких задач, далее решения комбинируются, и мы получаем отсортированный массив данных. Время выполнения сортировки во всех случаях равно o (N\*logN).

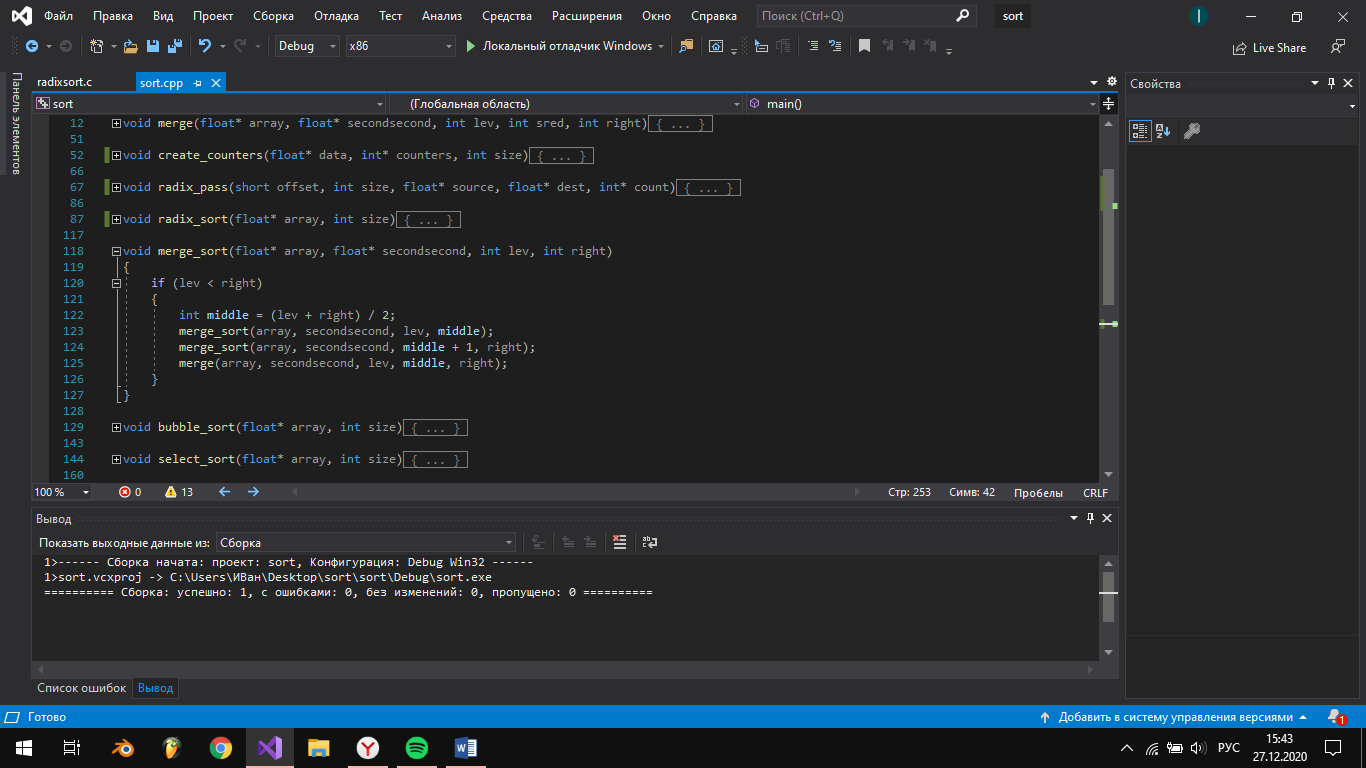


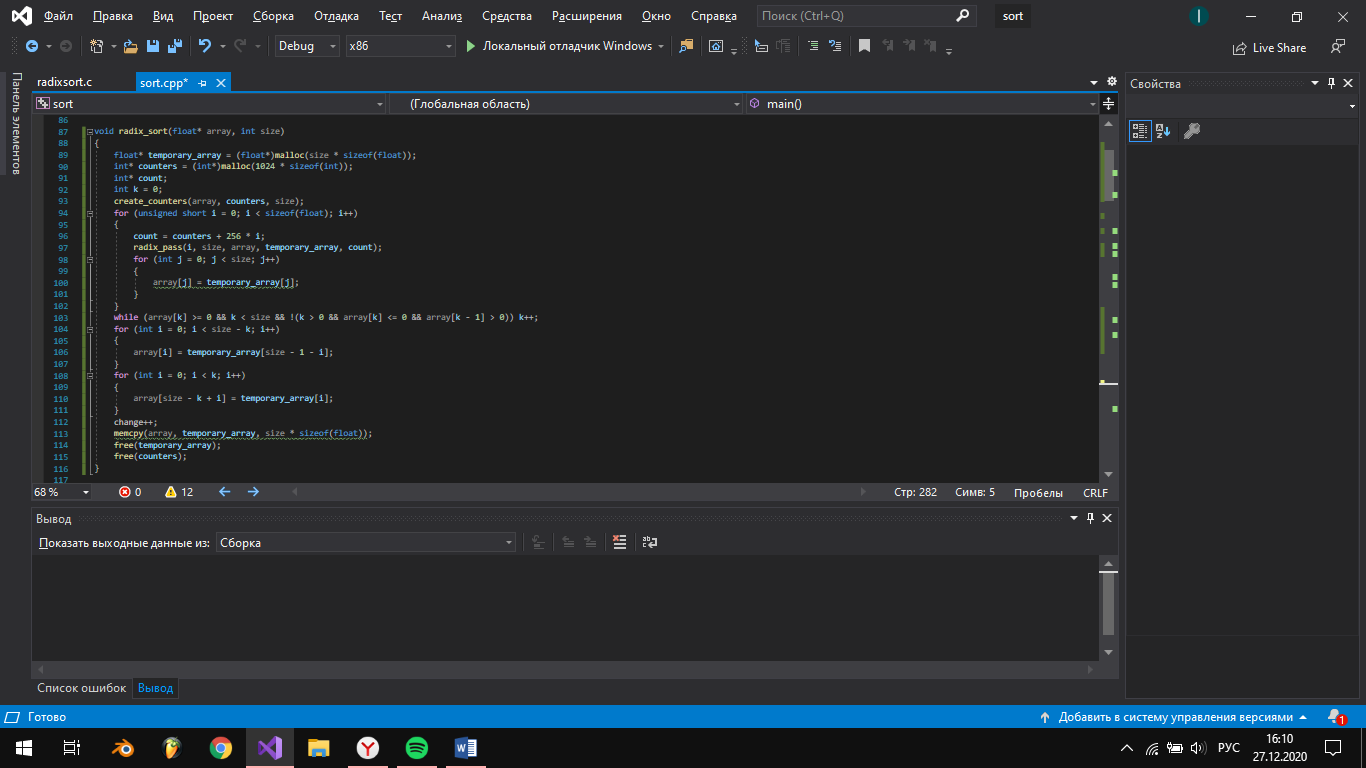
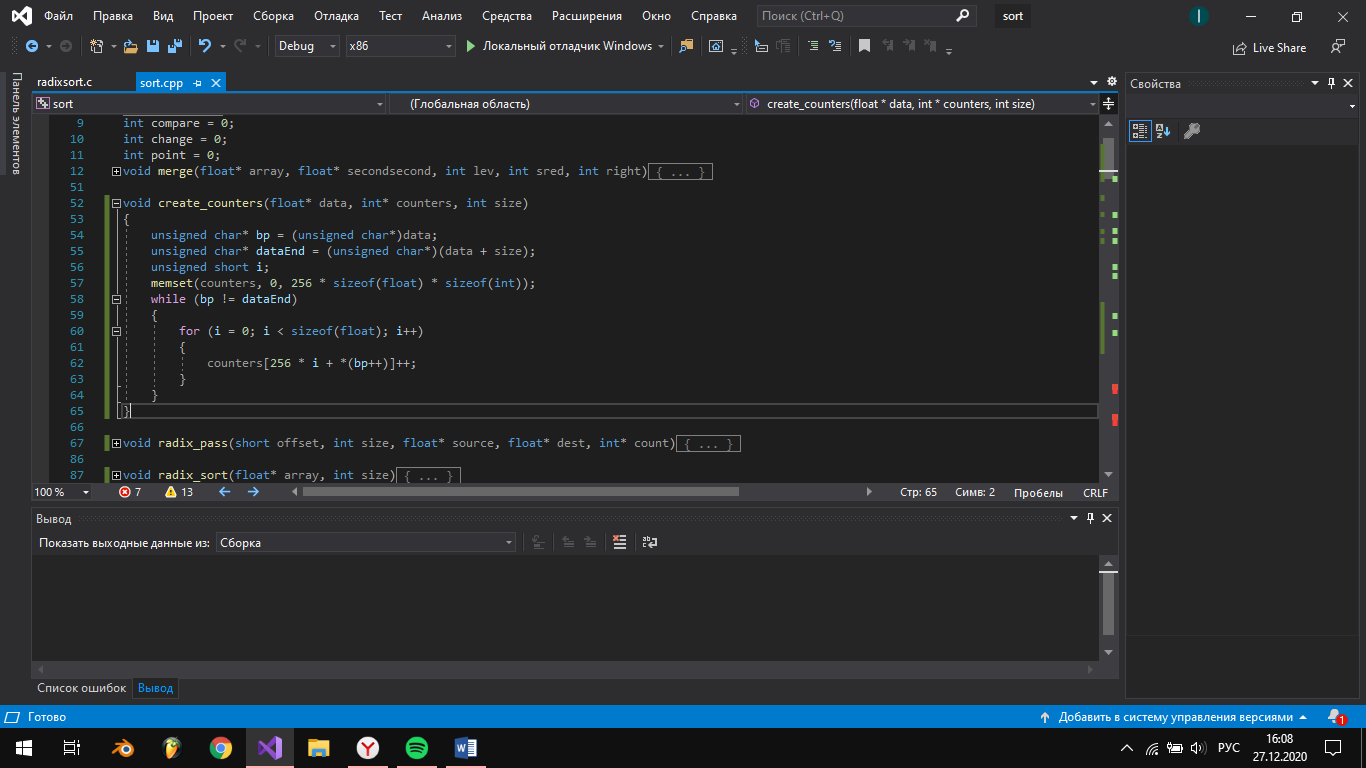
**Реализация:**

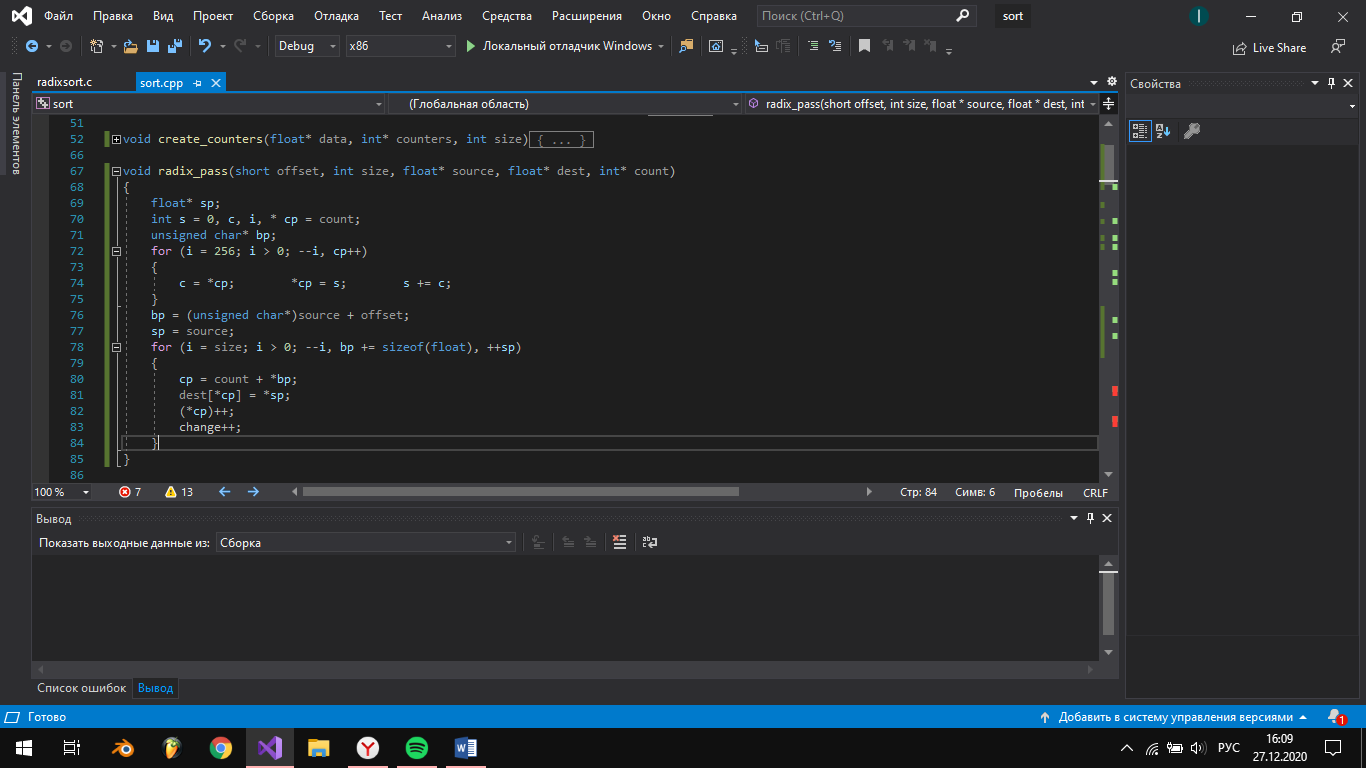
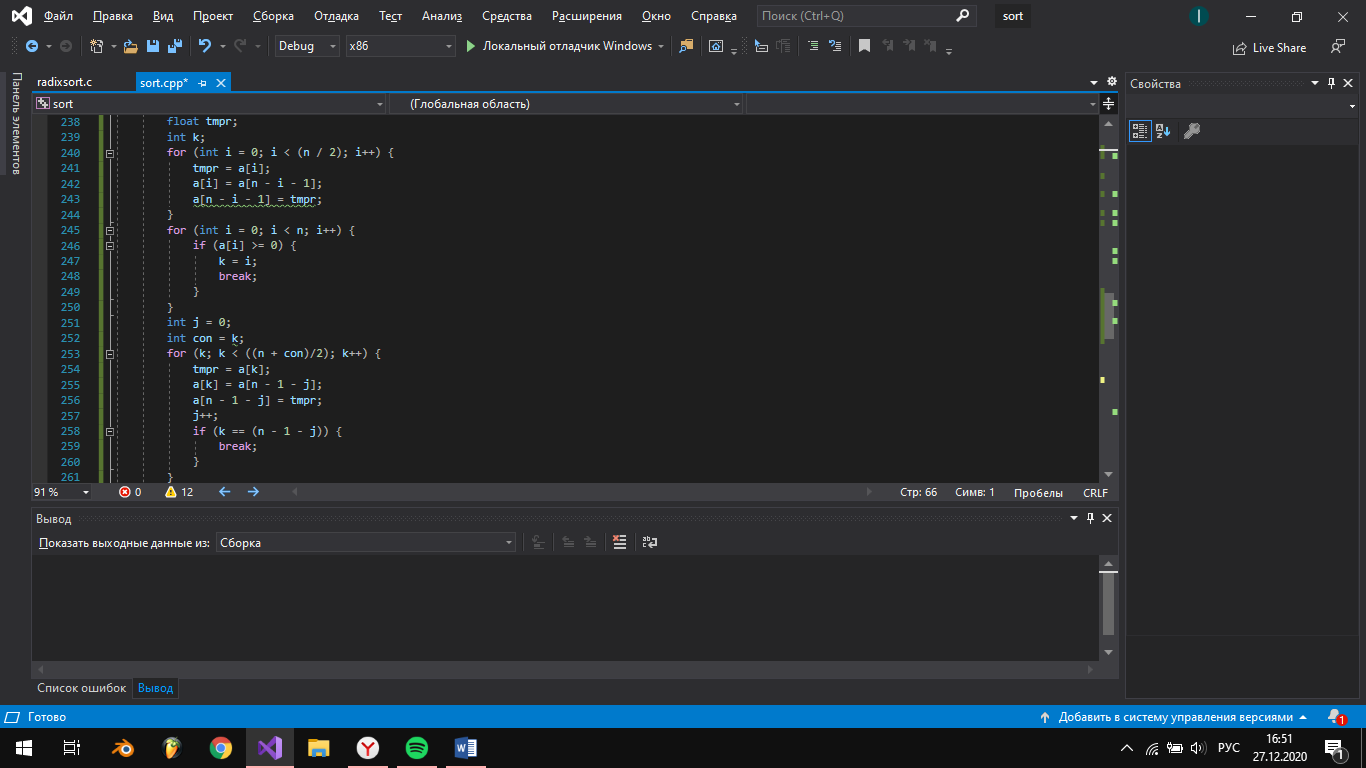
* Массив разбивается на две (практически) одинаковых части, каждая часть также рекурсивно разбивается на два одинаковых массива;
* Когда длина массива становится наименьшей (1 элемент), то вызывается функция merge;
* Данная функция сортирует подмассив, и затем производит слияние двух массивов. В зависимости от введенных данных, отсортированный массив может оказаться либо до среднего элемента, либо после.

1. **Поразрядная сортировка.**

Вид линейной сортировки. Это означает, что время выполнения сортировки равно o (N).

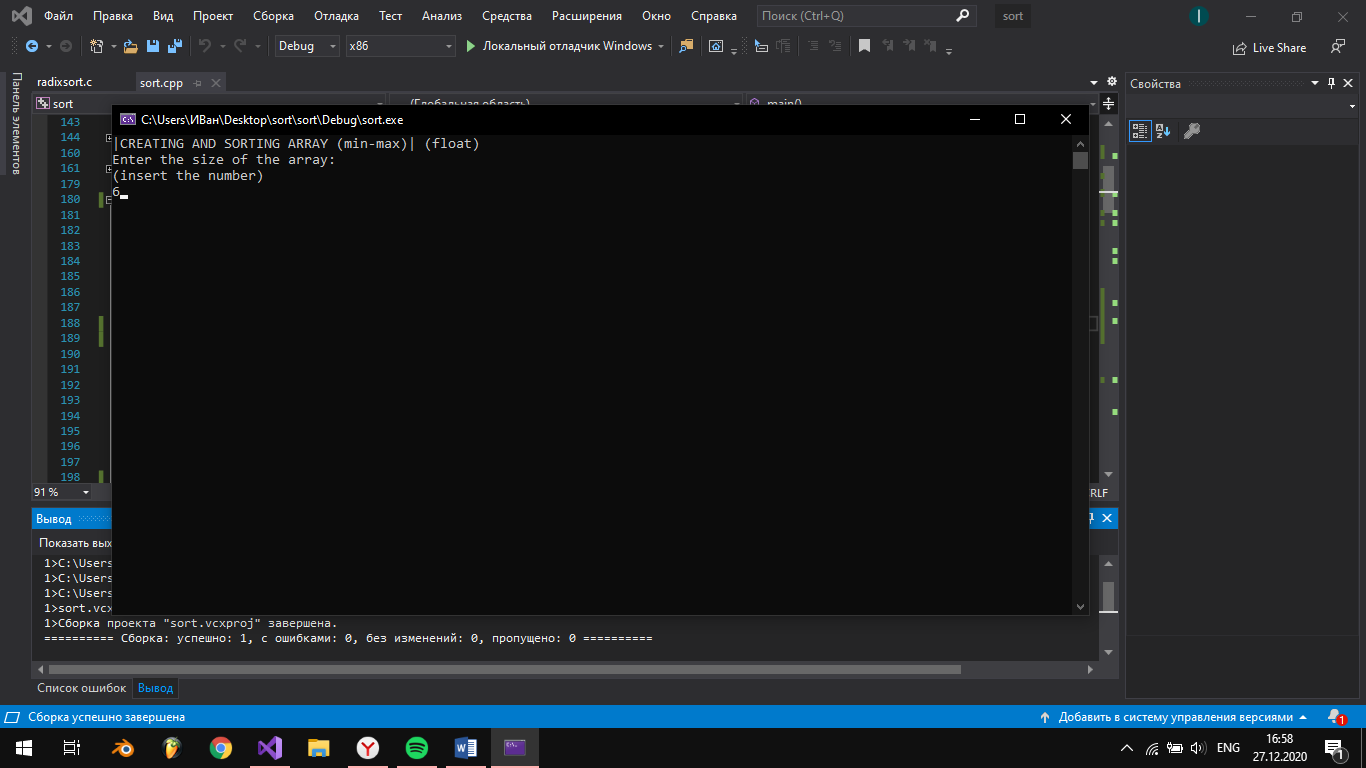
**Реализация:**

* ****Берется последняя цифра числа, подсчитывается количество таких чисел, которые также оканчиваются на эту же самую цифру;
* Далее резервируется место для каждой цифры в массиве;
* Данные числа записываются в выделенное место со сдвигом указателя;
* Данный алгоритм применяется устойчиво ко всем разрядам;
* В итоге мы получаем отсортированный массив, где первая часть представляет собой отсортированные по возрастанию положительные числа, а вторая часть отсортированные по убыванию отрицательные числа (так как отрицательные числа, по месту, которое они занимают в памяти, больше всех положительных).
* Для решения этой проблемы воспользуемся указанным алгоритмом, который первоначально меняет местами элементы массива так, что отрицательные числа будут расположены в порядке возрастания в начале массива, а положительные элементы в порядке убывания.
* Находим индекс первого положительного элемента и сортируем по возрастанию положительную часть массива.

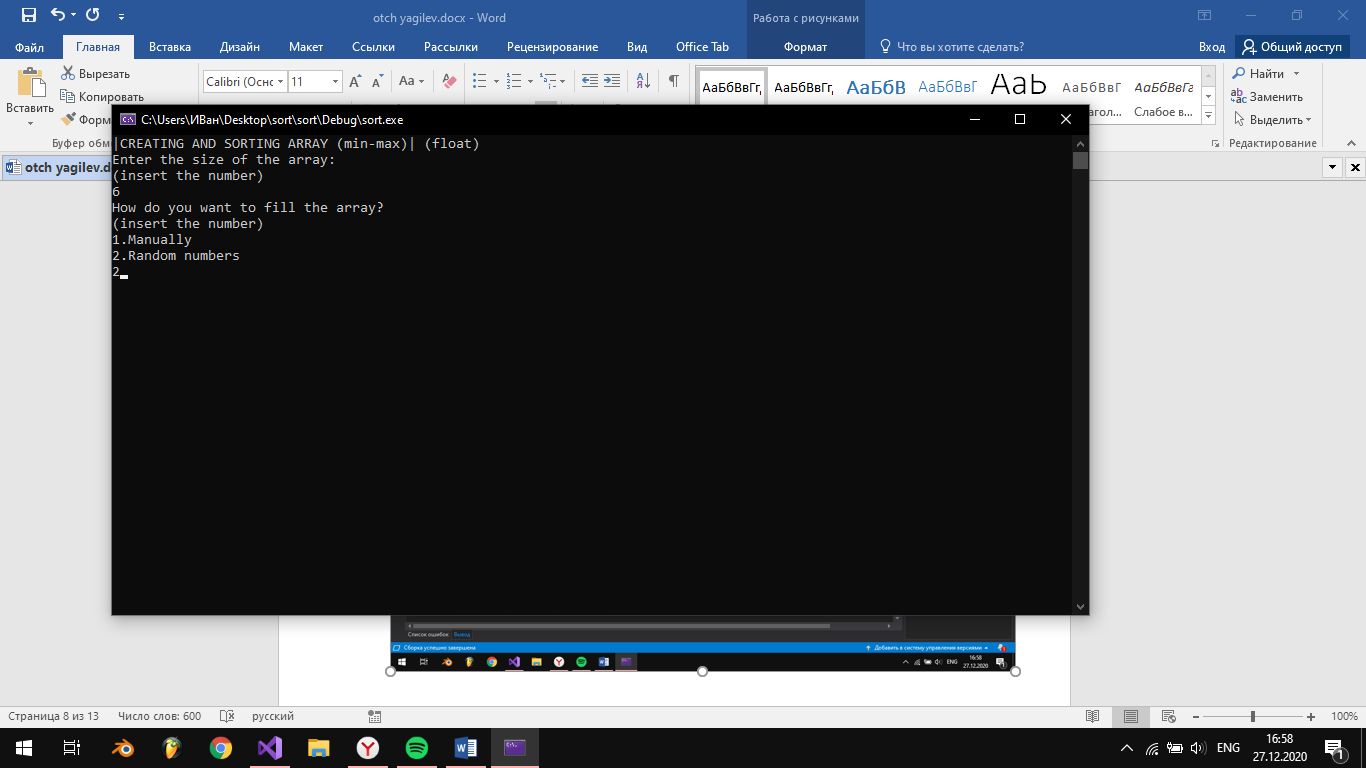


# Руководство пользователя

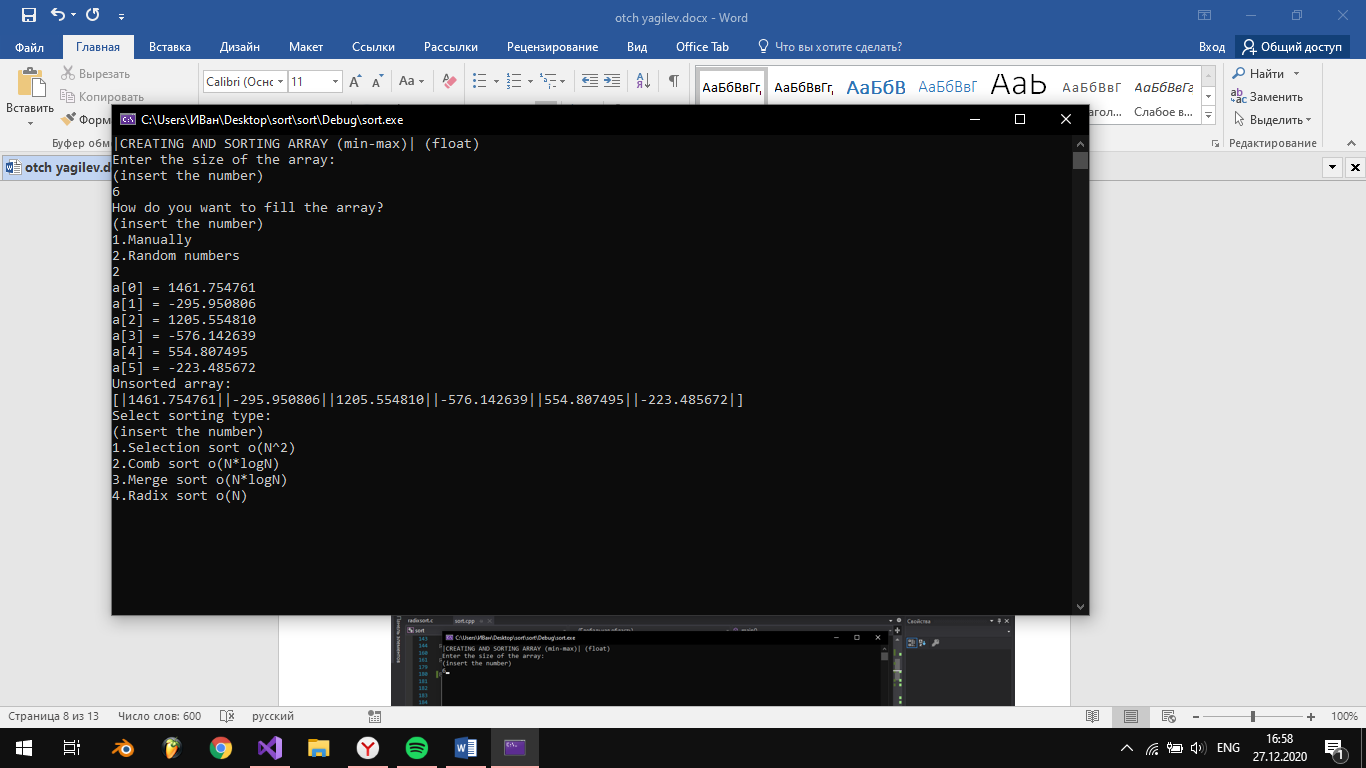
При запуске программы пользователь будет наблюдать, собственно, само название программы. Также пользователю будет предложено ввести размер массива.



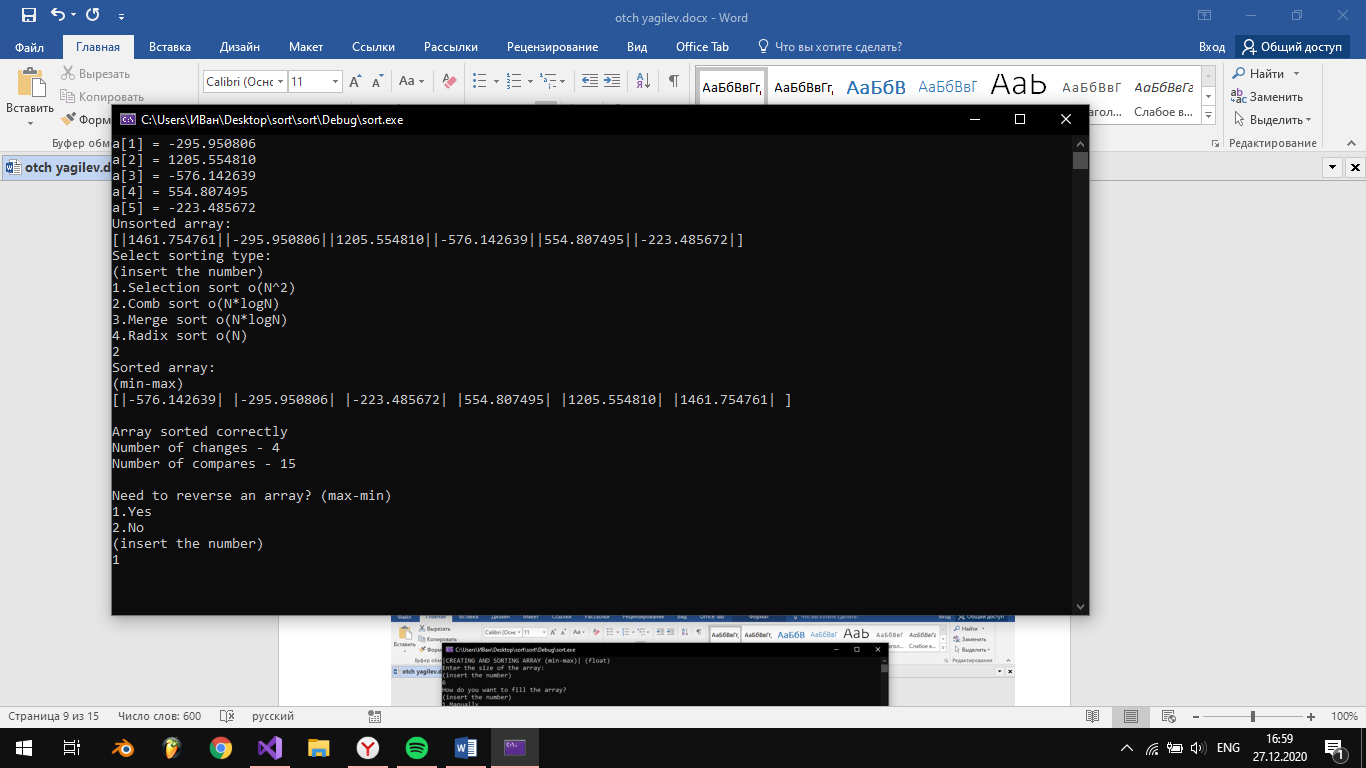
После ввода размера массива, у пользователя появляется возможность выбора способа заполнения массива. Ввод вручную, или же заполнение массива случайными числами.

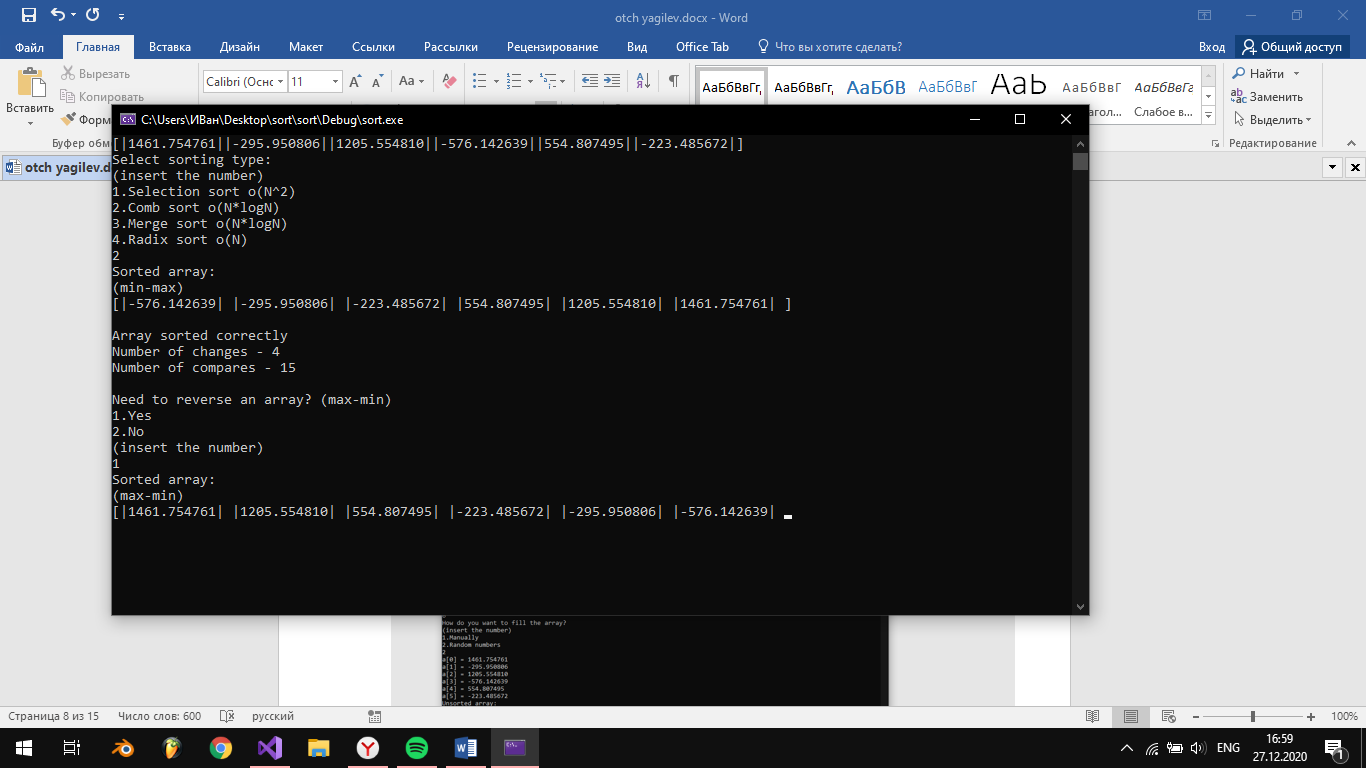


После выбора способа заполнения, пользователю будет дана возможность ввести вручную каждый элемент массива, или же ему просто выведется сгенерированное значение каждого элемента. Далее будет выведен неотсортированный массив и появится возможность выбора вида сортировки.

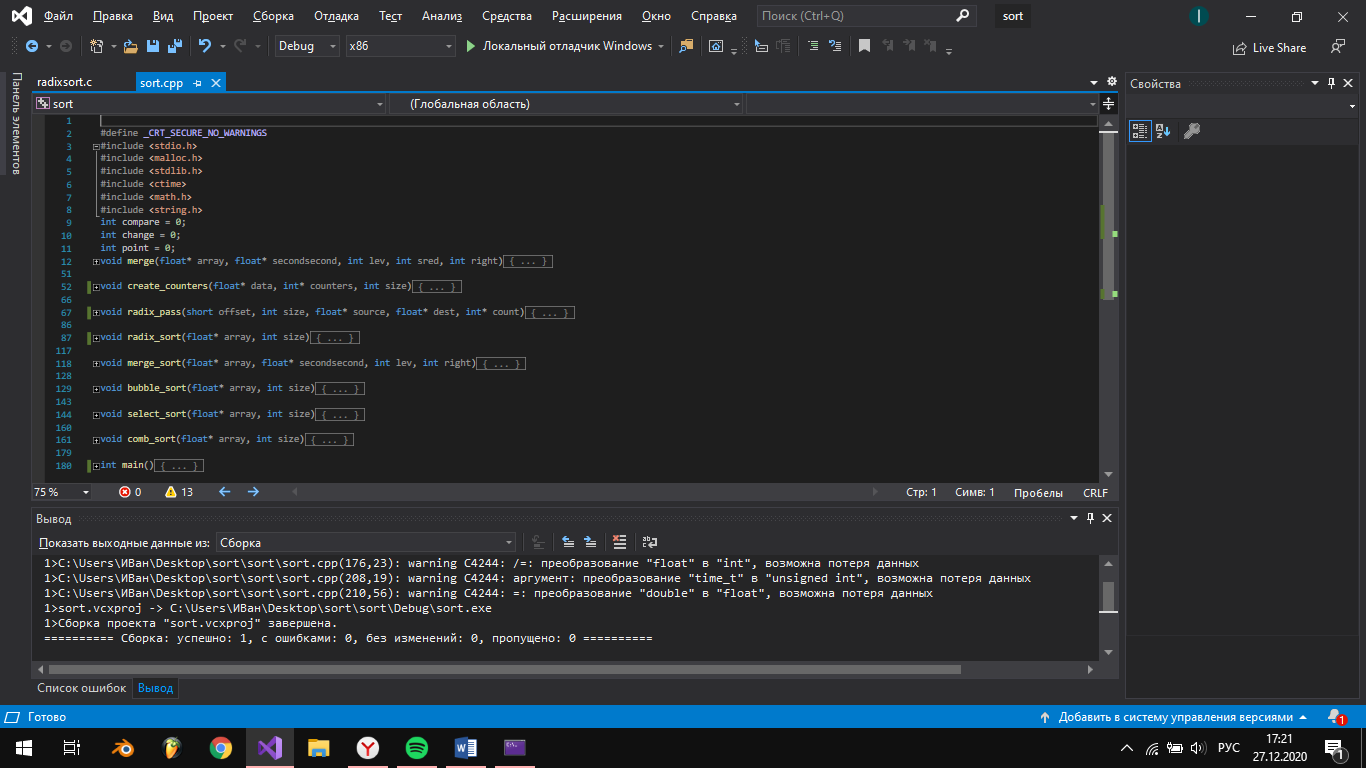


Вне зависимости от выбора пользователя, ему будет выведен отсортированный по возрастанию массив, корректно ли был отсортирован массив, число перестановок и сравнений (отличие будет в реализации сортировки и, естественно, в зависимости от выбора сортировки, будет отличаться число перестановок и сравнений).



Также пользователю будет предложено отсортировать массив по убыванию. В зависимости от ответа будет выведен отсортированный п убыванию массив, или же выполнение программы просто закончится.

# Описание программной реализации

Проект состоит из четырёх основных блоков:

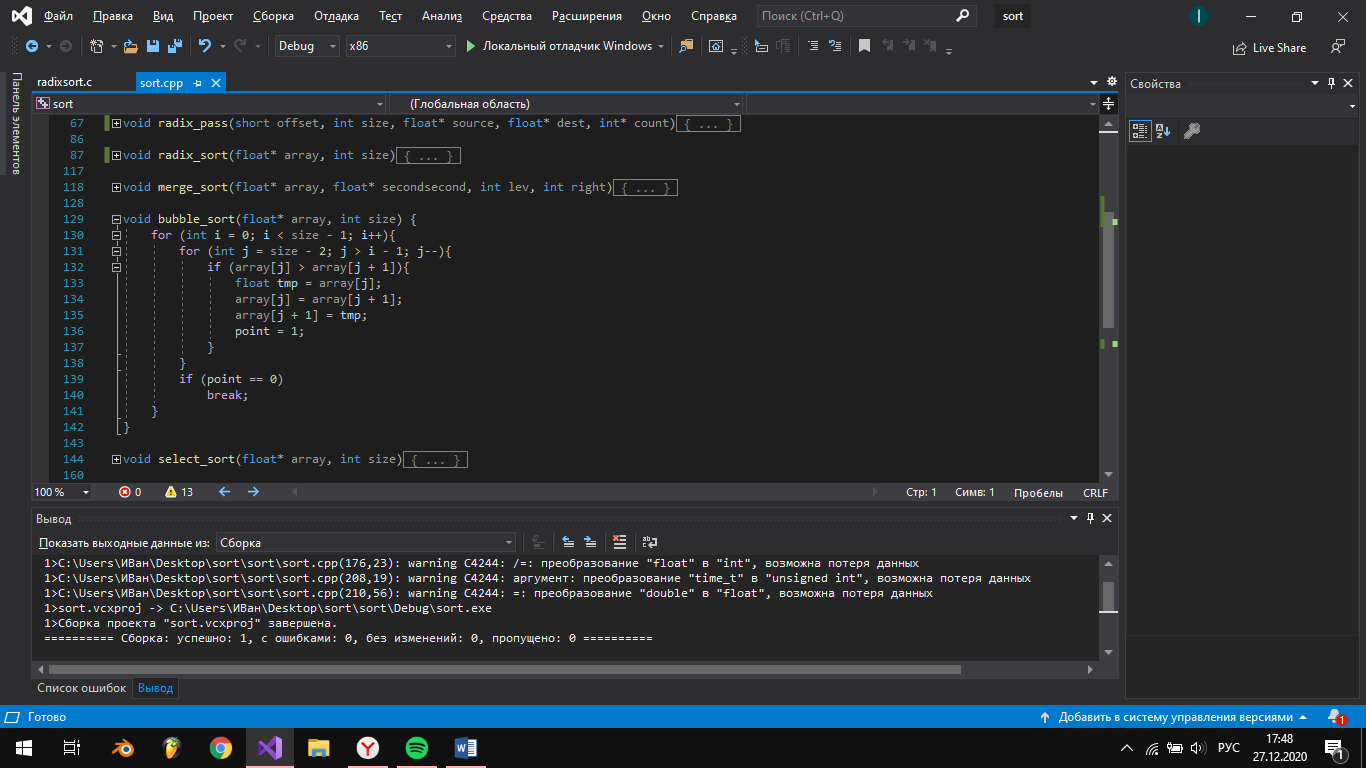
1. **Объявление библиотек:**

* **stdio**.**h** — стандартный заголовочный файл ввода-вывода).
* **malloc.h** – для функции с соответствующим названием, выделение динамической памяти.
* **stdlib.h** - заголовочный файл стандартной библиотеки языка Си, который содержит в себе функции, занимающиеся выделением памяти, контролем процесса выполнения программы, преобразованием типов и другие.
* **ctime.h** – работа с датой и временем (srand).
* **math.h** – функция возведения в степень, используется для создания отрицательных случайных чисел.
* string.h – работа со строками.

1. **Инициализация переменных** – количество перестановок, сравнений, константа, использующаяся для проверки корректности сортировки.
2. **Функции сортировки** (описаны выше, кроме сортировки пузырьком – проверка корректности)
3. **Основная часть кода (main) –** интерфейс, реализация выбора заполнения массива, выбора сортировки и другой функционал, описанный ранее.

# Подтверждение корректности

Для подтверждения корректности в программе используется вызов функции сортировки пузырьком, который делает проход по «отсортированному» массиву и выясняет, действительно ли массив отсортирован. Для этого испльзуется переменная point, которая имеет значение 1, если перестановка в массиве все-таки произошла (некорректность сортировки). Если же после одного прохода переменная имеет значение 0, то это означает корректность сортировки и позволяет нам выполнить выход из цикла.



# Результаты экспериментов

1. **Сортировка выбором:**

**Сложность алгоритма:** во всех случаях o (N^2).

Лучший случай – отсортированный массив. Средний случай – случайно заполненный массив. Худший случай - массив, отсортированный в обратном порядке.

Зависимость количества сравнений и перестановок от количества элементов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Количество элементов | 100 | 300 | 500 | 800 | 1000 |
| Лучший случай | 4950/0 | 44850/0 | 124750/0 | 319600/0 | 499500/0 |
| Средний случай | 4950/315 | 44850/1350 | 124750/2355 | 319600/4168 | 499500/5501 |
| Худший случай | 4950/2460 | 44850/22500 | 124750/62382 | 319600/159308 | 499500/247689 |

1. **Сортировка расческой:**

**Сложность алгоритма:** лучший случай – o (N\*log N), худший случай – o (N^2), средний случай – o (N^2/2^p), где p – количество приращений.

Лучший случай – отсортированный массив. Средний случай – отсортированный в обратном порядке массив. Худший случай – случайное заполнение массива.

Зависимость количества сравнений и перестановок от количества элементов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Количество элементов | 100 | 300 | 500 | 800 | 1000 |
| Лучший случай | 1233/0 | 5144/0 | 9536/0 | 16835/0 | 22034/0 |
| Худший случай | 1233/236 | 5144/925 | 9536/1837 | 16835/3720 | 22034/4588 |
| Средний случай | 1233/111 | 5144/407 | 9536/688 | 16835/1183 | 22034/1499 |

1. **Сортировка слиянием:**

**Сложность алгоритма:** во всех случаях - o (N\*log N).

Лучший случай – отсортированный массив. Средний случай – случайное заполнение массива.

Зависимость количества сравнений и перестановок от количества элементов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Количество элементов | 100 | 300 | 500 | 800 | 1000 |
| Лучший случай | 347/672 | 1307/2488 | 2273/4488 | 4058/7776 | 5044/9976 |
| Средний случай | 561/672 | 2125/2488 | 3926/4488 | 6819/7776 | 8830/9976 |

1. **Поразрядная сортировка:**

**Сложность алгоритма:** во всех случаях o (N).

Расположение элементов в массиве никак не влияет на сложность сортировки, так как алгоритм работает с памятью.

Зависимость количества сравнений и перестановок от количества элементов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Количество элементов | 100 | 300 | 500 | 800 | 1000 |
| Общий случай | 0/800 | 0/2400 | 0/4000 | 0/6400 | 0/8000 |

# Заключение

Мы успешно реализовали четыре алгоритма сортировки для типа данных float, показали теоретическую сложность алгоритмов на результатах эксперимента. Убедились в корректности данных сортировок.

# Приложение

