### 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 382003-1 Назаров Н.А.

## Проверил:

### ассистент каф. МОСТ, Волокитин В.Д.

### Нижний Новгород 2020

**Содержание**

[Постановка задачи 3](#_bookmark0)

[Метод решения 4](#_bookmark1)

[Руководство пользователя 5](#_bookmark2)

[Описание программной реализации 6](#_bookmark3)

[Подтверждение корректности](#_bookmark4) 8

[Результаты экспериментов](#_bookmark5) 9

[Заключение](#_bookmark6) 11

[Приложение 12](#_bookmark7)

# Постановка задачи

# Передо мной стоит задача реализовать алгоритмы сортировки пузырьком, быстрой сортировки, сортировки слиянием и поразрядной. Реализовать данные алгоритмы необходимо для чисел типа double. При реализации нужно учесть подсчет сравнений и перестановок в каждом из алгоритмов сортировки. Так же нужно проверить выполнение алгоритмов для лучшего, среднего и худшего случая, чтобы показать их теоретическую сложность.

# **Метод решения**

1) Сортировка Пузырьком (BubbleSort)

Метод сортировки пузырьком заключается в последовательном сравнении и перестановки соседних элементов, если предшествующий оказывается больше последующего. В процессе выполнения данного алгоритма элементы с большими значениями оказываются в конце списка, а элементы с меньшими значениями постепенно перемещаются по направлению к началу списка.

2) Быстрая сортировка (QuickSort)

Метод быстрой сортировки заключается в следующем:

-Из исходного массива выбирается некоторый элемент, который принимается в качестве разделителя или опорного элемента.

-Все ключи, меньшие разделителя, располагаются до него, а все большие – после него.

-Перестановка элементов выполняется путём обмена местами ключей, которые необходимо переместить в другую часть массива.

-При этом обмениваются ключи, расположенные на большом расстоянии друг от друга и этим достигается наивысший эффект упорядочивания.

3) Сортировка слиянием (MergeSort)

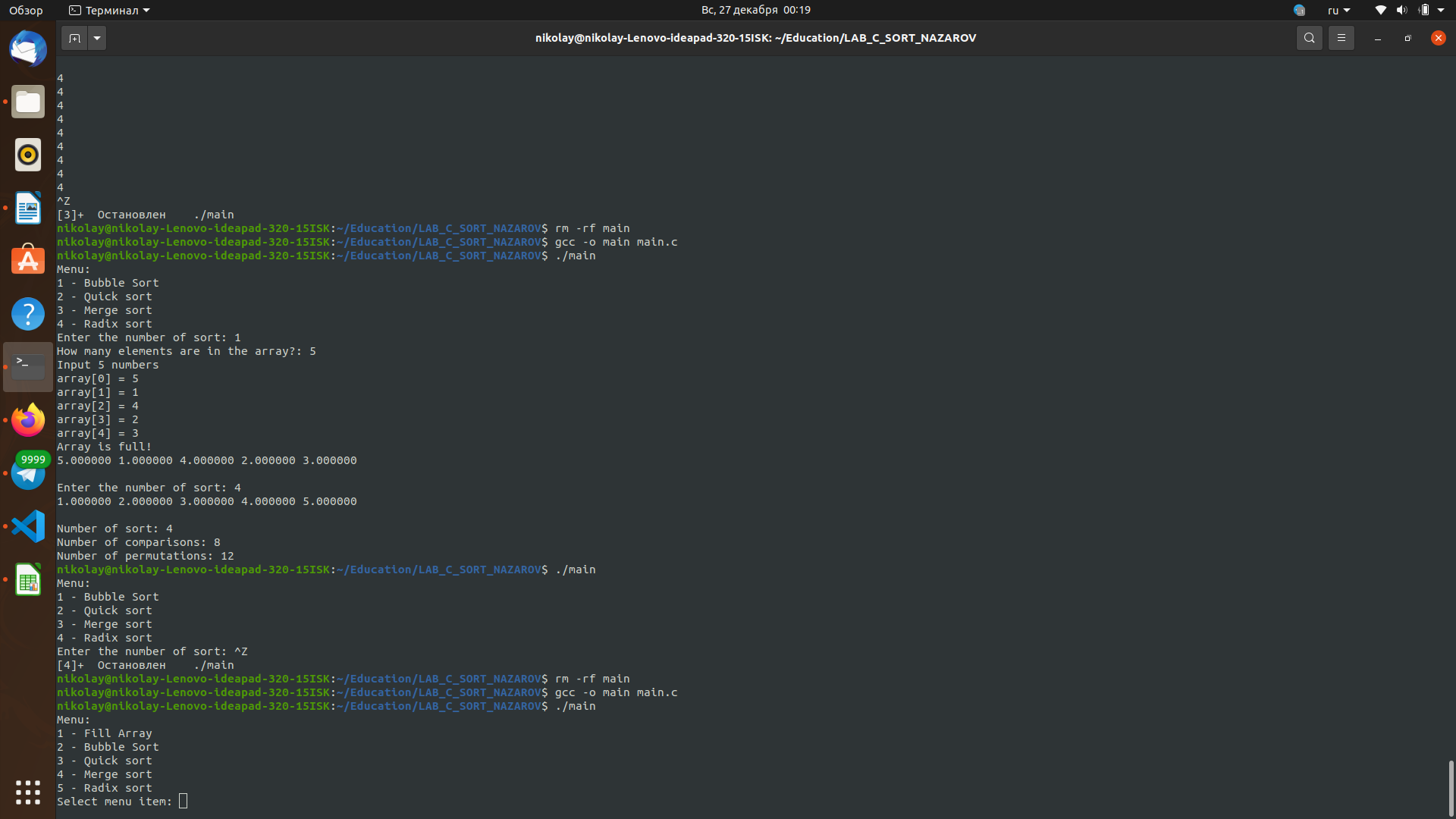
Основная идея сортировки слиянием — разделение массива на два подмассива, сортировка их тем же методом, и их объединение. Над массивами проводится тот же алгоритм. Массивы делятся до тех пор, пока их длина больше единицы. Массив длинны 1 считается отсортированным.  
При слиянии берем меньший элемент из двух первых элементов подмассивов и записываем в результирующий массив.

4) Поразрядная сортировка (RadixSort)

Суть поразрядной сортировки заключается в том, что мы сортируем числа по наименьшему разряду, потом переходим на следующий разряд, на котором производим такие же действия, это повторяется до тех пор пока алгоритм не отсортирует старший разряд, после чего массив можно считать отсортированнным.

**Руководство пользователя**

Для взаимодействия с программой пользователю доступно меню:

При выборе первого пункта меню, пользователь может ввести количество элементов в массиве, после чего может вводить элементы последовательно.

После ввода всех элементов пользователь должен выбрать нужный алгоритм сортировки из меню, записать его номер в консоль, в ответ он получит отсортированный массив и количество перестановок и сравнений.

**Описание программной реализации**

Вся программа состоит из одного файла main.c

Описание функций:  
- inputArray() первым аргументом принимает указатель на массив, вторым размер массива. Обрабатывает введенные пользователем данные и вносит их в массив.  
- fillArrayRandomNumber() первым аргументом принимает указатель на массив, вторым размер массива. Заполняет массив псевдослучайными числами.  
- fillArrayInvert() первым аргументом принимает указатель на массив, вторым размер массива. Заполняет массив числами от size до единицы.  
- fillArrayEasy() первым аргументом принимает указатель на массив, вторым размер массива. Заполняет массив числами от нуля до size.  
- showArray() первым аргументом принимает указатель на массив, вторым размер массива. Выводит элементы массива.  
- compare() функция для сортировки qsort. Сравнивает два числа.  
- checkSort() первым аргументом принимает указатель на массив, вторым его размер. Создает копию массива и сортирует ее с помощью библиотечной функции qsort. Сравнивает массивы поэлементно и выводит результат(совпадают ли массивы или нет)  
- swap() Оба аргумента принимают указатели на переменные тип double. Меняет значения переменных местами.  
- BubbleSort() первым аргументом принимает указатель на массив, вторым размер массива. Сортирует массив с помощью алгоритма сортировки пузырьком.  
- partition() первым аргументом принимает указатель на подмассив сортируемого массива, вторым его левую границу, третьим его правую границу. Является вспомогательной функцией функции quickSort.  
- quickSort() первым аргументом принимает указатель на массив, вторым его левую границу, третьим правую. Сортирует массив с помощью алгоритма быстрой сортировки.  
- merge() первым аргументом принимает указатель на массив слияния,вторым и третьим отсортированные подмассивы, четвертым левую границу массива слияния, пятым правую массива слияния. Составляет из двух отсортированных подмассивов новый отсортированный массив. Вспомогательная функция функции mergeSort.   
- mergeSort() первым аргументом принимает указатель на массив, вторым его левую границу, третьим правую. Сортирует массив с помощью алгоритма сортировки слиянием.  
- create\_counters() первым аргументом принимает указатель на массив, вторым указатель на массив подсчета, третьим размер первого массива. Вспомогательная функция функции radixSort. Считает значение каждого байта у элемента.  
- radix\_pass() первым аргументом принимает номер байта от 0 до 3, вторым длину массива, третьим указатель на исходный массив, четвертым указатель на дополнительный массив, пятым аргументом принимает указатель на элемент массива подсчета, с которого функция начнет сортировку по этому разряду. Вспомогательная функция функции radixSort.  
- radixSort() первым аргументом принимает указатель на массив, вторым его размер. Сортирует массив с помощью алгоритма сортировки подсчетом.

**Подтверждение корректности**

Для подтверждения корректности выполнения каждого алгоритма сортировки использую функцию checkArray. Функция принимает отсортированный массив, создает его копию, которую сортирует с помощью библиотечной функции qsort, сравнивает поэлементно два получившихся массива и в случае совпадения массивов, выдает сообщение, что сортировка произведена корректно.

**Результаты экспериментов**

1) Сортировка пузырьком:

Таблица с зависимостью перестановок от количества элементов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Длина массива | 10 | 50 | 100 | 500 | 1000 |
| Лучший случай | 9 | 49 | 96 | 499 | 999 |
| Средний случай | 24 | 627 | 2483 | 62527 | 249901 |
| Худший случай | 45 | 1225 | 4950 | 124750 | 499500 |

Лучший случай — отсортированный массив О(n)

Средний случай — рандомный массив O(n^2)

Худший случай — обратно отсортированный O(n^2)

2) Быстрая сортировка:

Таблица с зависимостью перестановок/сравнений от количества элементов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Длина массива | 10 | 50 | 100 | 500 | 1000 |
| Лучший случай | 0/25 | 0/237 | 0/573 | 0/3989 | 0/8977 |
| Средний случай | 9/26 | 59/298 | 151/744 | 1024/4673 | 2393/9523 |
| Худший случай | - | - | - | - | - |

Лучший случай — отсортированный массив О(n\*log(n))

Средний случай — рандомный массив О(n\*log(n))

3) Сортировка слиянием:

Таблица с зависимостью перестановок/сравнений от количества элементов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Длина массива | 10 | 50 | 100 | 500 | 1000 |
| Лучший случай | 34/15 | 286/133 | 672/316 | 4488/2216 | 9976/4932 |
| Средний случай | 34/24 | 286/222 | 672/557 | 4488/3934 | 9976/8819 |
| Худший случай | - | - | - | - | - |

Лучший случай — отсортированный массив О(n\*log(n))

Средний случай — рандомный массив О(n\*log(n))

4) Сортировка поразрядная:

Таблица с зависимостью перестановок/сравнений от количества элементов

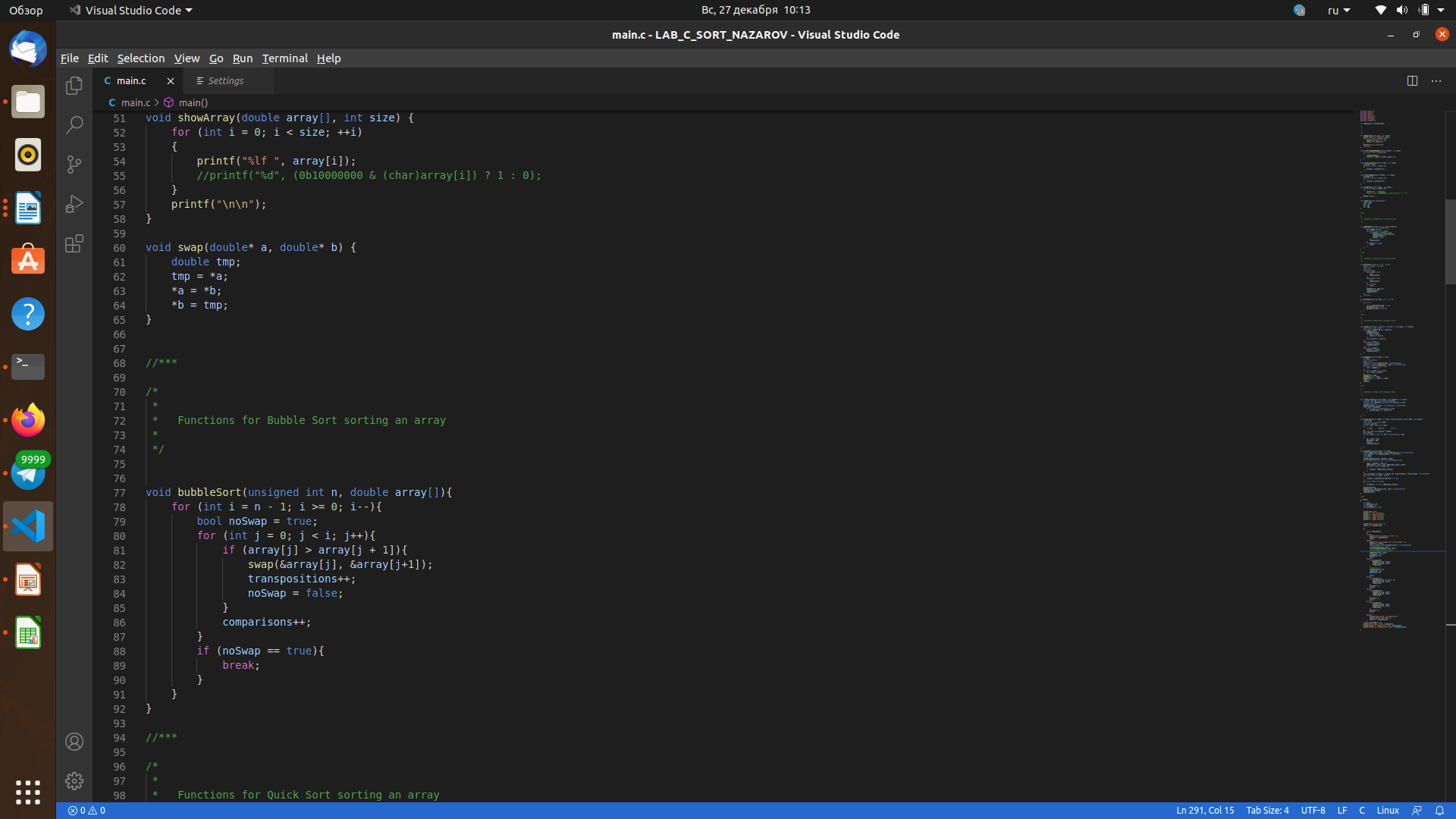
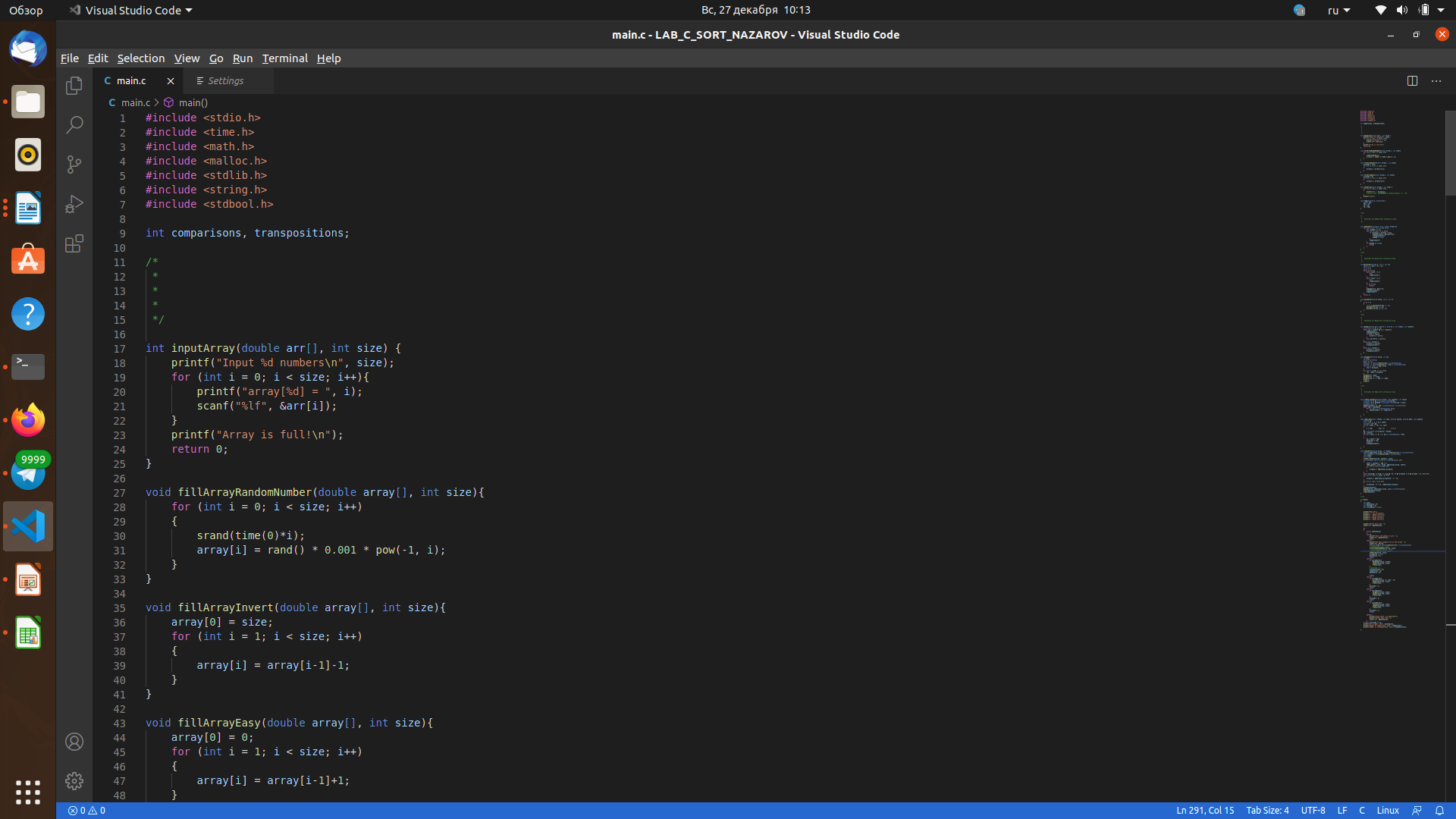
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Длина массива | 10 | 50 | 100 | 500 | 1000 |
| Лучший случай | 50/0 | 250/0 | 500/0 | 2500/0 | 5000/0 |
| Средний случай | - | - | - | - | - |
| Худший случай | - | - | - | - | - |

Не зависит от входных данных, поэтому нет лучшего/среднего/худшего случая  
Сложность О(n)

**Заключение**

Поставленная задача выполнена. Реализовано несколько алгоритмов сортировки. При выполнении тестов, подтвердил, что сложность алгоритма сходится с теоритической сложностью.

**Приложение**

****

