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 3821Б1-ПМ2

Фамилия И.О.

Токарев Александр Алексеевич

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 7](#_Toc26962564)

[Описание программной реализации 8](#_Toc26962565)

[Подтверждение корректности 10](#_Toc26962566)

[Результаты экспериментов 11](#_Toc26962567)

[Заключение 15](#_Toc26962568)

[Приложение 16](#_Toc26962569)

# Постановка задачи

Задание заключалось в следующем:

**1.**Реализовать следующие сортировки для типа данных float :

1) Сортировка вставками (англ. Insertion Sort)

2) Быстрая сортировка (англ. Quick Sort)

3) Сортировка слиянием (англ. Merge Sort)

4) Поразрядная сортировка (англ. Radix Sort)

И проверить их корректность.

**2.** Замерить число перестановок и количество сравнений. выполненное при сортировке.

**3.** Поставить эксперименты так, чтобы показать теоретическую сложность алгоритмов, отразить её в отчёте.

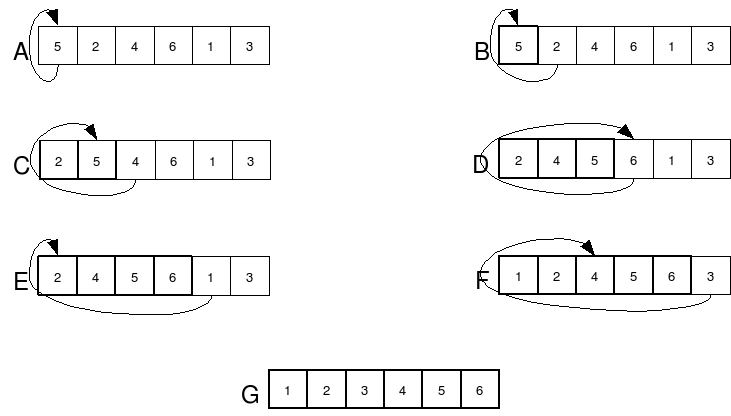
# Метод решения

Подробное описание алгоритмов для каждой из сортировок.

1. Сортировка вставками (англ. Insertion Sort)

В данном алгоритме элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Ниже представлена пошаговая иллюстрация:



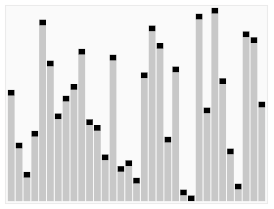
1. Быстрая сортировка (англ. Quick Sort)

Один из самых быстрых известных универсальных алгоритмов сортировки массивов.

Общая идея алгоритма состоит в следующем:

* Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.
* Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
* Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.
* В представленной реализации в качестве опорного будет выбран последний элемент

Ниже представлена анимация:



1. Сортировка слиянием (англ. Merge Sort)

Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью [рекурсивного вызова](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F) или непосредственно, если их размер достаточно мал. Наконец, их [решения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87) комбинируются, и получается решение исходной задачи.

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый [массив](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

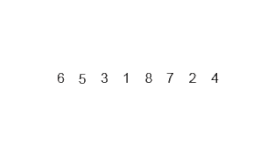
1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.  
Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по возрастанию подмассива. Тогда:

3.2. Слияние двух подмассивов в третий результирующий массив.  
На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.

3.3. «Прицепление» остатка.  
Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

Ниже представлена анимация:



1. Поразрядная сортировка (англ. Radix Sort)

Алгоритм пригоден для сортировки любых объектов, запись которых можно поделить на «разряды», содержащие сравнимые значения.

Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

# Руководство пользователя

Каждый алгоритм сортировки представлен в виде отдельной программы.

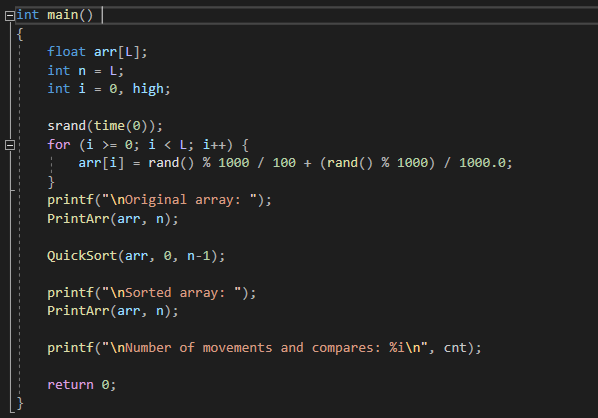
Данные программы, при запуске, автоматически генерируют массив из 10 элементов (используя функцию rand).

После выполнения на экран выводится два массива, первый изначальный массив, а второй – этот же массив, но уже отсортированный.

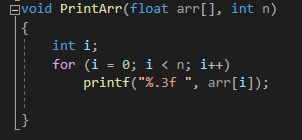
Также программа выводит количество операций, т.е. сумму сравнений и перестановок, выполненных в ходе сортировки.

# Описание программной реализации

* Структура проекта, какие есть файлы и что в них содержится
* Каждая сортировка выполнена в виде отдельной программы, файлы каждой сортировки лежат в отдельной папке. Программы состоят из отдельных функций, общими для каждой программы будут функции main и PrintArr.
* Функция main генерирует массив (в случае с Radix Sort два массива) из 10 элементов с помощью rand (зависящей от текущего времени), далее вызывает функции печати и саму функцию сортировки.



* PrintArr имеет следующий вид и служит соответственно для печати массива.



Далее подробнее о реализации каждой из сортировок:

1. **Insertion Sort** реализована в виде одной функции.

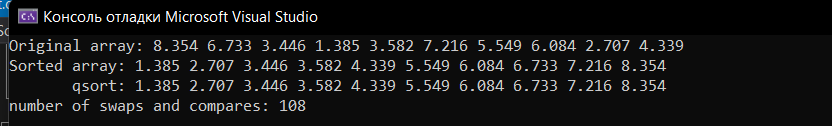
В неё поступает массив и его размер (число элементов), далее выполняется сортировка вставками, алгоритм работы которой описан выше.

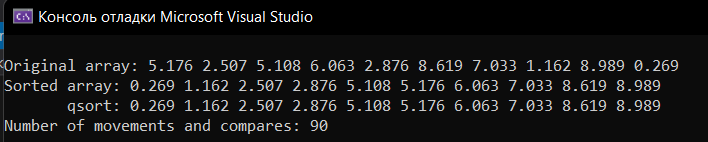
1. **Quick Sort** реализована в виде одноимённой функции, в неё поступает массив и индексы крайнего левого и крайнего правого элемента, она вызывает partition (разбиение массива,  перераспределение элементов в массиве таким образом, что элементы, меньшие опорного, помещаются перед ним, а большие или равные - после.), передаёт ей массив и индексы элементов, после работы **partition** возвращает значение индекса опорного элемента после перестановки, оно присваивается в pi, далее Quick Sort рекурсивно вызывает саму себя, передавая массив и индексы начала и конца подмассивов слева и справа от опорного элемента.Partition использует **swap** - обмен значений элементов через третью переменную .
2. **Merge Sort** реализована в виде одноимённой функции, в неё поступает массив и индексы крайних элементов, далее вычисляется индекс среднего элемента и функция вызывает саму себя, чтобы разделить массив на части (рекурсивное разбиение массивов на меньшие происходит до тех пор, пока размер массива не достигнет единицы) а затем вызывается функция **merge**, в которую передаётся массив и индексы крайних и среднего элемента (merge - слияние упорядоченных массивов, массив длинной в 1 элемент можно считать упорядоченным)
3. **Radix Sort** реализована в виде 4 функций, в саму **radixSort** поступают два одинаковых массива и их размер, далее она вызывает вспомогательную **createCounters**, передавая в неё массив и его размер, **createCounters** имеет возвращаемое значение. Далее в рамках цикла вызывается **radixPass**, в которую передаются оба массива и их размер, переменная цикла и count. В итоге работы цикла получится массив, состоящий из двух подмассивов (положительные числа и отрицательные в обратном порядке) Чтобы это исправить radixSort вызывает модификацию **signedRadixLastsort**, которая инвертирует подмассив отрицательных чисел и поставит их перед положительными. В модификацию передаются оба массива и их размер.

# Подтверждение корректности

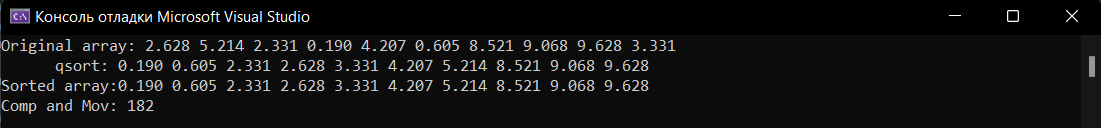
Для подтверждения корректности в программах использовалась сортировка из стандартной библиотеки <stdlib.h> qsort. В каждой программе подавалось два одинаковых массива, один сортировался с помощью qsort, другой с помощью соответствующей сортировки. Далее результаты работы сравнивались.

Для Insertion Sort:

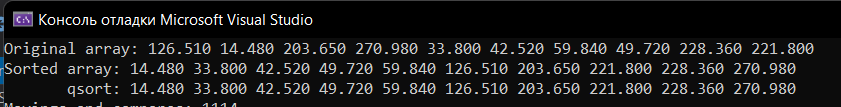
 Для Quick Sort:



Для Merge Sort:



Для Radix Sort:



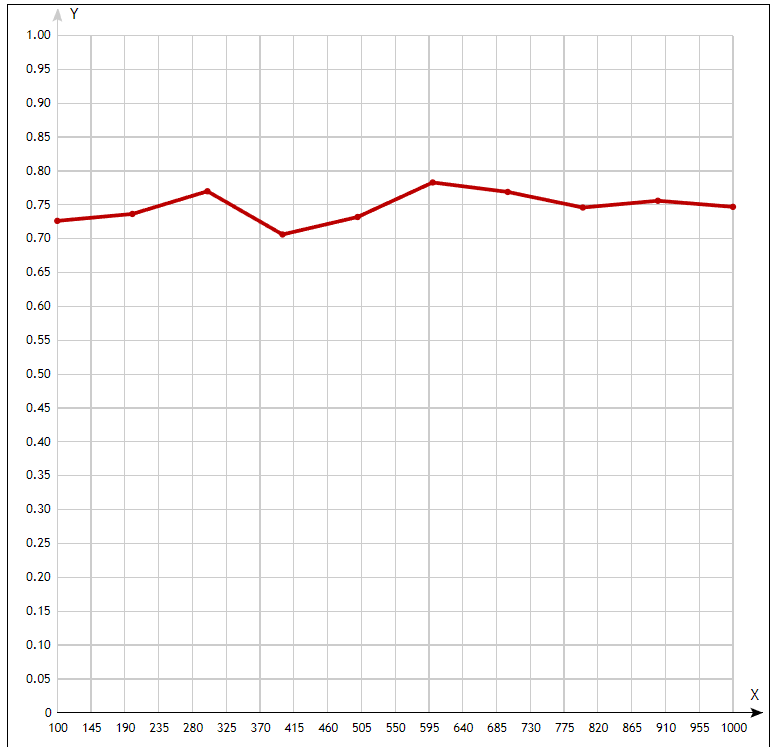
Таким образом, мы видим, что в каждой программе результаты сортировок совпадают, что подтверждает корректность работы реализованных сортировок.

# Результаты экспериментов

По данным экспериментов видно, что сортировки отличаются сложностью. Для доказательства сложности построим графики.

Докажем теоретическую сложность сортировок:

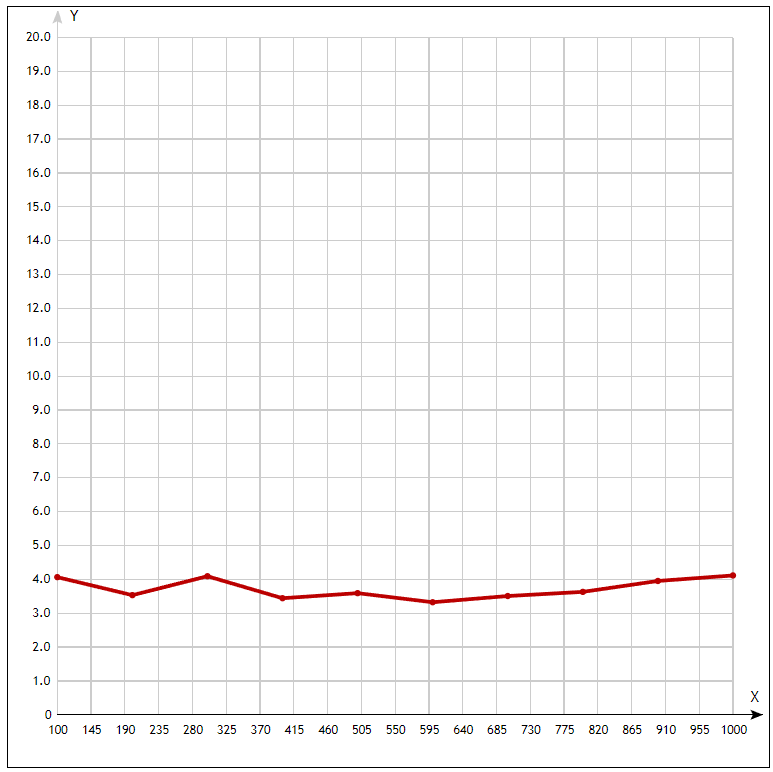
1. Insertion Sort – простая сортировка, её вычислительная сложность должна составлять О(*n*2), Для подтверждения построим график, где числу элементов (n) массива (от 100 до 1000 с шагом 100) соответствует (сумма числа сравнений и присвоений) / (*n*2)



Каждая точка графика находится в пределах от 0,7 до 0,8, Учитывая, что они вычисляются как (число операций)/(число элементов в квадрате), можно сделать вывод, что алгоритм в среднем работает за 0.75\* *n*2 операций, при вычислении сложности принято отбрасывать константу, таким образом, получается доказать, что алгоритм имеет квадратичную сложность.

1. Quick Sort

Данная сортировка соответствует своему названию, её сложность должна составлять в среднем O(n\*log(n)), однако она имеет худший случай O(*n*2). Построим график, аналогичный предыдущему, с учётом, что числу элементов (n) массива (от 100 до 1000 с шагом 100) соответствует (сумма числа сравнений и присвоений) / (n\*log(n))



Данный график показывает, что алгоритм, в среднем, действительно имеет рост сложности n\*log(n) (без учёта константного коэффициента), точки меняют значение в пределах от 3,2 до 4,2, в среднем алгоритм срабатывает за 3.6\*n\*log(n) операций.

1. Merge Sort

Данная сортировка также, как и предыдущая относится к категории быстрых, однако отличается своей стабильностью, так как не имеет худшего и лучшего случая и работает всегда за O(n\*log(n)) операций.

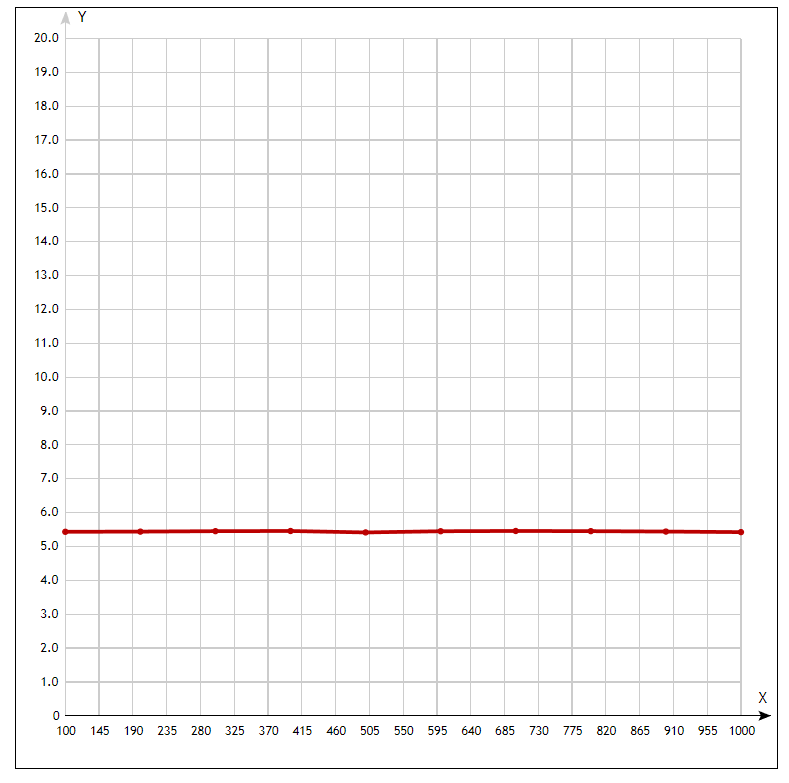


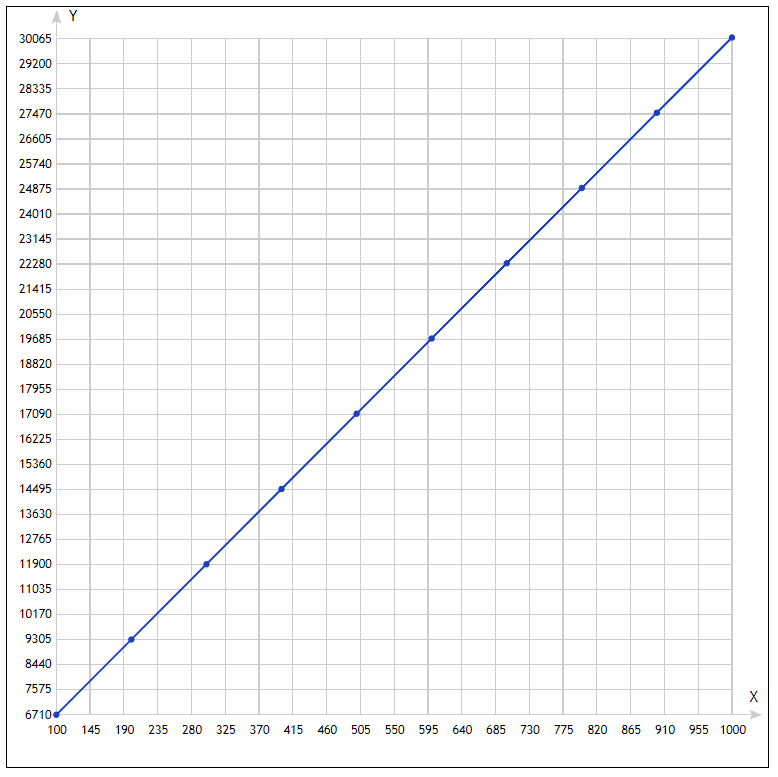
График (построенный аналогично предыдущему) подтверждает рост сложности n\*log(n), стоит отметить, что значения точек практически не отличаются друг от друга (Это связано с отсутствием худшего и лучшего случая), они приблизительно равняются 5.4,

Таким образом сортировка работает за 5.4\*n\*log(n) операций

4)Radix Sort

Данная сортировка имеет линейный рост сложности O(w\*n), где w – вес элемента в битах, что делает её самой быстрой, по крайней мере для массивов с большим числом элементов. Стоит отметить, что число операций зависит от типа входных данных. На этот раз просто построим график зависимости числа операций, от числа элементов.

Из графика видно, что сортировка действительно имеет линейную сложность. (график прямая)



# Заключение

В ходе лабораторной работы были реализованы следующие сортировки для типа данных float :

1) Сортировка вставками (англ. Insertion Sort)

2) Быстрая сортировка (англ. Quick Sort)

3) Сортировка слиянием (англ. Merge Sort)

4) Поразрядная сортировка (англ. Radix Sort)

Алгоритм их работы описан в пункте **Метод Решения,** а программная реализация в пункте **Описание программной реализации**.

Каждая сортировка успешно прошла проверку на корректность.

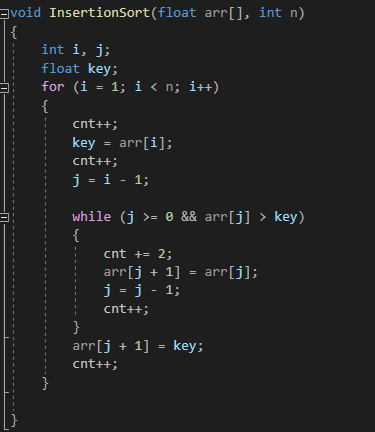
Массивы были корректно отсортированы, без потери или добавления элементов.

Для каждой сортировки был произведён замер числа перестановок и количества сравнений при размерах массива от 100 до 1000 элементов с шагом в 100.

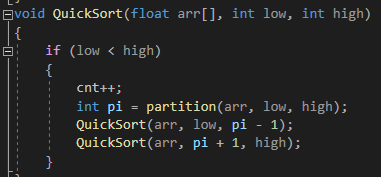
Далее на основе этих данных строились графики, и была доказана теоретическая сложность алгоритмов.

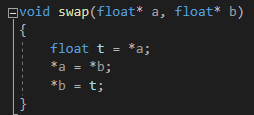
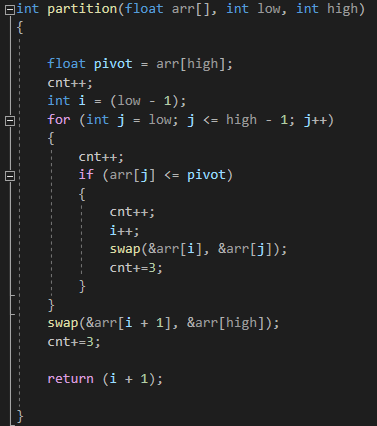
# Приложение

Код Insertion Sort:

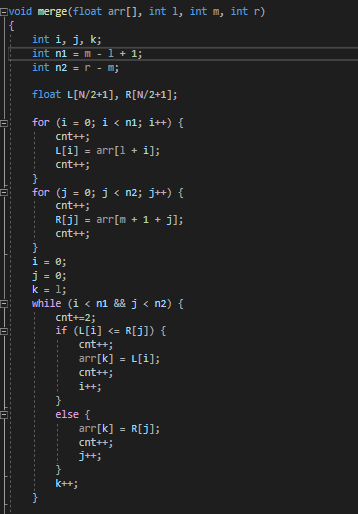
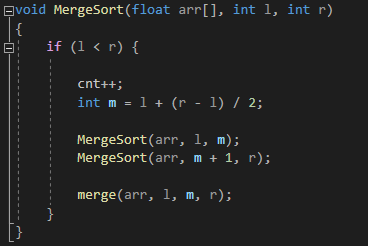


Код Quick Sort:





Код Merge Sort:



Код Radix Sort

