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**

**«Сортировка одномерных массивов в C»**

**Выполнил**:

студент группы 382003-1

Боев Р.А.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2019

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 6](#_Toc26962564)

[Описание программной реализации 7](#_Toc26962565)

[Подтверждение корректности 11](#_Toc26962566)

[Результаты экспериментов 14](#_Toc26962567)

[Заключение 15](#_Toc26962568)

[Приложение 16](#_Toc26962569)

# Постановка задачи

Изучить алгоритмы сортировки массивов и научиться использовать их при обработке данных. Освоить основные приемы использования массивов, методов доступа к элементам массивов, их реорганизации и модификации.

В качестве практической проблемы, требующей решения, рассматривается известная задача сортировки (упорядочивания) массива в порядке возрастания его элементов. При решении этой задачи требуется исходный массив, содержащий произвольные числовые данные, преобразовать к виду, когда каждый элемент массива находится перед другим элементом этого массива, если его значение меньше, чем значение сравниваемого элемента.

Мы рассмотрим сортировку пузырьком, Хоара, слиянием и поразрядную на примере сортировки данных типа double.

# Метод решения

*Сортировка пузырьком*

Начиная с первого элемента, сравниваем 2 соседних элемента, и если они расположены не по возрастанию, то меняем их местами. За 1-й проход 1-й элемент встаёт на своё место, за 2-й проход – 2-й элемент и т.д. Когда (N-1)-ый элемент встанет на своё место, последнему элементу ничего не останется кроме как тоже встать на своё место. Всего N-1 проходов.

В 1-й проход будет выполнено N-1 сравнение, в каждый следующий на 1 меньше. Всего будет 1 + 2 + 3+…N-1 = N(N-1)/2 сравнения и в худшем случае такое же количество перестановок. Сложность этого алгоритма - O(n2)

*Сортировка Хоара*

Выбираем в массиве некоторый элемент, который будем называть *опорным элементом*.   
Операция разделения массива: реорганизуем массив таким образом, чтобы все элементы, меньшие или равные опорному элементу, оказались слева от него, а все элементы, большие опорного — справа от него.

Два индекса — l и r, приравниваются к минимальному и максимальному индексу разделяемого массива соответственно. Вычисляется индекс опорного элемента m. Индекс l последовательно увеличивается до m до тех пор, пока l-й элемент не превысит опорный. Индекс r последовательно уменьшается до m до тех пор, пока r-й элемент не окажется меньше либо равен опорному. Если r = l — найдена середина массива — операция разделения закончена, оба индекса указывают на опорный элемент. Если l < r — найденную пару элементов нужно обменять местами и продолжить операцию разделения с тех значений l и r, которые были достигнуты. Следует учесть, что если какая-либо граница (l или r) дошла до опорного элемента, то при обмене значение m изменяется на r-й или l-й элемент соответственно. Рекурсивно упорядочиваем подмассивы, лежащие слева и справа от опорного элемента. Базой рекурсии являются наборы, состоящие из одного или двух элементов. Первый возвращается в исходном виде, во втором, при необходимости, сортировка сводится к перестановке двух элементов. Все такие отрезки уже упорядочены в процессе разделения.

Операция разделения массива на две части относительно опорного элемента занимает время О(log2n). Операции разделения, проделываемые на одной глубине рекурсии, обрабатывают разные части исходного массива, размер которого постоянен, суммарно на каждом уровне рекурсии потребуется также О(n). В среднем сложность алгоритма O(nlogn), однако в плохих случаях он может деградировать до О(n2).

*Сортировка слиянием*

Сортировка слиянием использует стратегию разделяй и властвуй. Выбирается элемент-разделитель, а оставшиеся элементы делятся на две равные части, а затем алгоритм рекурсивно вызывает сам себя для их сортировки. Потом отсортированные половины сливаются в комбинированный отсортированный список.

Кроме сортируемых массивов, начального и конечного индексов, алгоритм также берет в качестве параметра дополнительный массив, который использует для объединения отсортированных половин

Код начинает работу с проверки общего кол-ва элементов в части массива. Если элемент один, значит, массив отсортирован и алгоритм прекращает выполняться. Если элементов несколько, алгоритм рассчитывает индекс того, что находится в середине части, и рекурсивно вызывает сам себя для сортировки половин. Затем отсортированные половины объединяются. Они пересматриваются в цикле, при этом меньший элемент копируется в рабочий массив. Если одна половина пуста, алгоритм копирует остальные элементы из другой половины. На завершающем этапе объединенные элементы из рабочего массива переносятся в исходный массив.

Можно объединить сортируемые половины и без рабочего массива, но это более сложный и медленный процесс, поэтому программисты такой массив используют крайне редко.

Глубина рекурсии – logn. На каждом уровне элементы сливаются в сумме в n элементов. (на втором уровне 2 массива по n/2, на 3-м – 4 по n/4 и т.д.). Поэтому сложность O(nlogn).

*Поразрядная сортировка*

Особенность поразрядной сортировки в том, что элементы непосредственно между собой не сравниваются. Сначала элементы сортируются по своему последнему разряду, затем следующему и т.д. до первого разряда.

На каждой своей итерации алгоритм включает 2 этапа: распределение и сборка. Скорость работы алгоритма определяется тем, что количество проходов по всем элементам исходного списка равна **максимальному количеству разрядов** в элементе, т.е. разрядов сортируемого ключа. Если мы сортируем числа, то количество разрядов - будет равно количеству десятичных разрядов этого числа, например: 27 - количество разрядов 2, 487 - количество разрядов 3, число 9 - 1 разряд и т.д.

Перед сортировкой необходимо определить 2 величины:

1) **width** - максимальное количество разрядов в сортируемых величинах, по-другому - количество разрядов в самом длинном ключе.

2) **range** - количество возможных значений одного разряда ключа(сортируемого элемента). Т.е. для десятичных чисел очевидно имеем - 10, для 16-тиричных - 16

Создаются range вспомогательных списков - "карманов", т.е. на каждое возможное значение разряда элемента - по карману, т.е. по списку. Далее - первый этап **распределение** по карманам и на первом проходе элементы исходной последовательности помещаются в эти карманы (списки) по их младшему разряду, т.е. по самому правому числу. Какой этот самый младший разряд у элемента, в такой карман этот элемент и помещается.

В классической поразрядной сортировке на каждом проходе, т.е. в пределах каждого разряда - элементы сортировались путем буквального разделения по каждому разряду, т.е. мы получали последовательности элементов, в каждой из которых содержались т.н. "одинаковые" элементы (идиентичны по конкретному текущему разряду). Очевидно, что для того, чтобы получить единую последовательность элементов отсортированных в пределах одного этого разряда - надо просто один за другой, по-порядку соединить все эти последовательности между собой.

Подход, о котором идет речь здесь - отличается тем, что на каждом проходе элементы исходной последовательности сортируются по конкретному текущему разряду **не с помощью разделения по карманам**, а с помощью подсчета элементов, которые меньше индекса некоторого дополнительного массива счетчиков. Т.е. никаких уже 2 этапов: распределения и сборки. Теперь можно сказать - только **подсчет** и **вставка** элементов в нужное место.

Подробнее. Те же основные итерации по каждому разряду максимально-разрядного числа, от младшего к старшему разряду. В пределах каждой итерации - составляем последовательность(назовем ее **positions**) из значений разрядов каждого элемента нашей последовательности, чтобы только с этими значениями на данной итерации и работать (сами числа нам не нужны конечно - только текущие их разряды). Далее составляем искомый массив счетчиков - **count**, размером [range](https://www.codelab.ru/task/4/#vals). Каждое его значение будет определять количество элементов из **positions**, которые меньше индекса данного элемента массива **count**. После того, как такой массив будет составлен - мы будем знать где на самом деле на данной итерации должен находится каждый элемент из **positions**, чтобы последний был отсортирован, поскольку для каждого элемента **positions** мы знаем количество элементов меньших чем он и поэтому встравляем его на соответствующее место.

# Руководство пользователя

*Сортировка пузырьком*

Пользователь должен открыть файл “Bubble.exe”

Ввести целое число от 1 до 100 (кол-во элементов массива).

После того как массив отсортирован, ввести любое целое число (это сделано, чтобы не происходил выход из программы сразу же после того, как она выведет упорядоченный массив).

*Сортировка Хоара*

Пользователь должен открыть файл “ConsoleApplication5.exe”

Ввести целое число от 1 до 100 (кол-во элементов массива).

После того как массив отсортирован, ввести любое целое число. (это сделано, чтобы не происходил выход из программы сразу же после того, как она выведет упорядоченный массив).

*Сортировка слиянием*

Пользователь должен открыть файл “mergesort.exe”

Ввести натуральное число (нужно для случайного заполнения массива).

После того как массив отсортирован, ввести любое целое число. (это сделано, чтобы не происходил выход из программы сразу же после того, как она выведет упорядоченный массив).

*Поразрядная сортировка*

Пользователь должен открыть файл “radsort.exe”

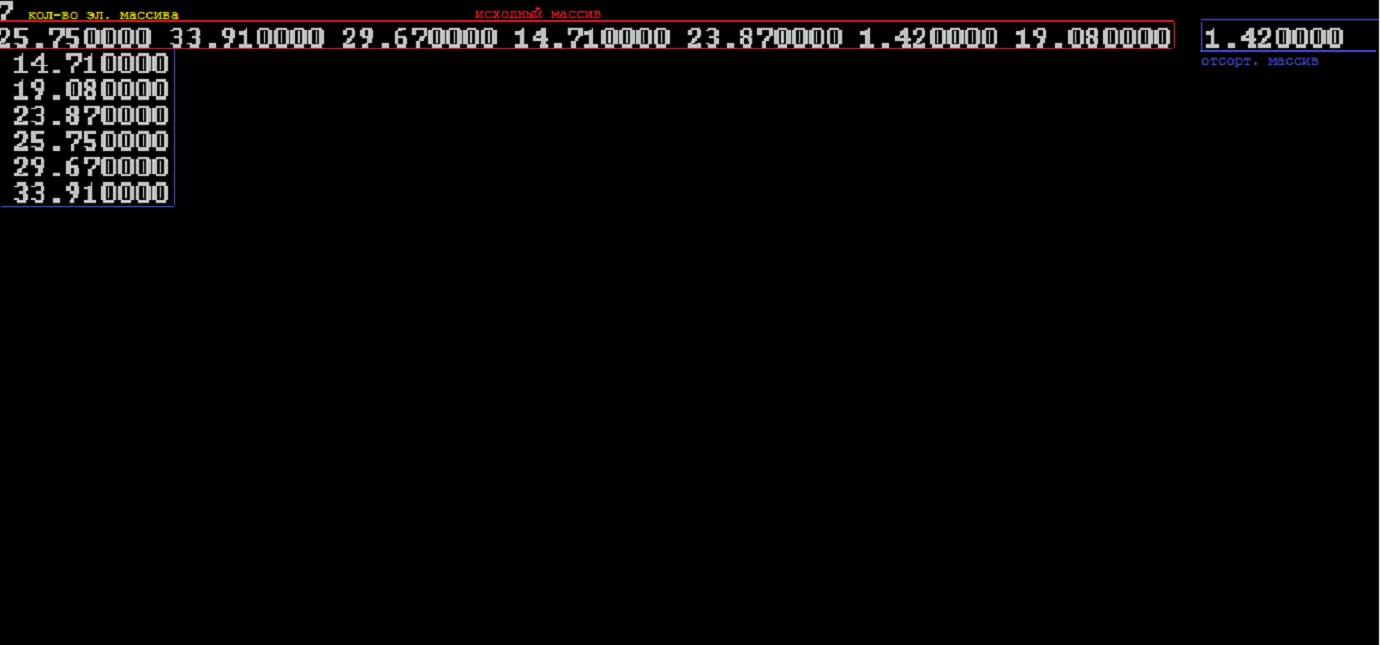
Ввести натуральное число (нужно для случайного заполнения массива).

После того как массив отсортирован, ввести любое целое число. (это сделано, чтобы не происходил выход из программы сразу же после того, как она выведет упорядоченный массив).

# Описание программной реализации

*Сортировка пузырьком*

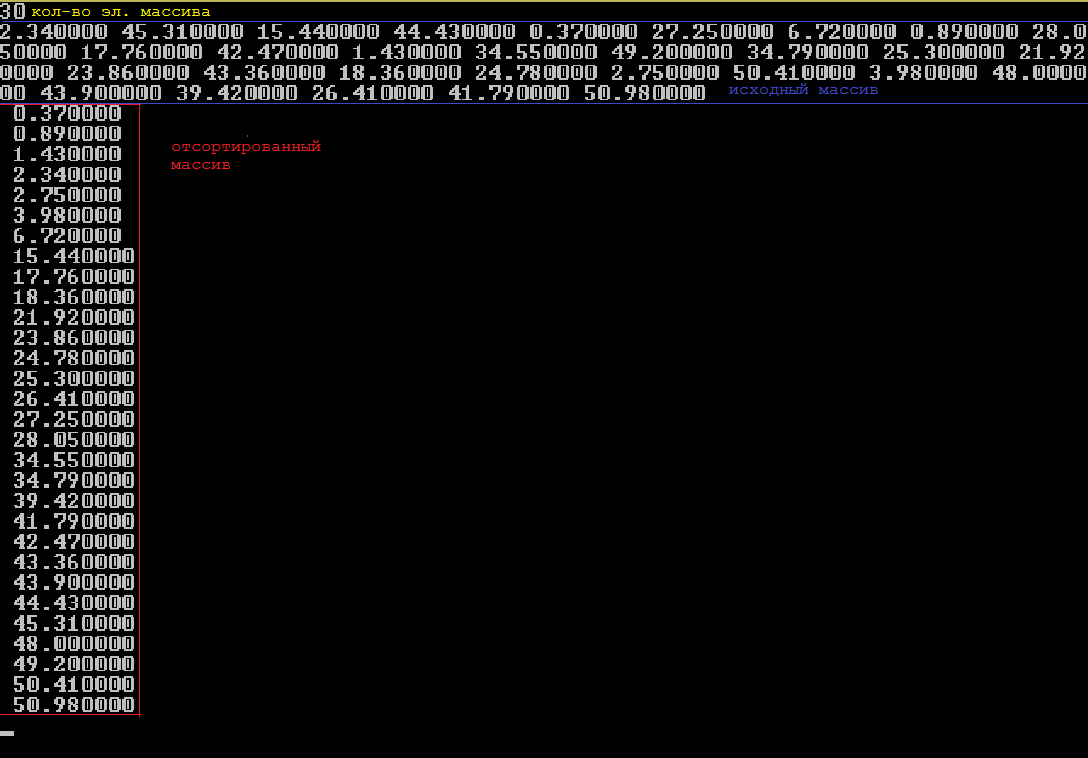
Программа запрашивает целое число от 1 до 100 – кол-во эл. массива. Заполняет массив случайными числами. Выводит исходный массив. Сортирует его. Выводит отсортированный массив. Запрашивает целое число. Выход из программы.



Программа запускается при помощи исполняемого файла “Bubble.exe”, при помощи которого запускается сама сортировка (вынесен отдельно для удобства проверки). Так же в папке “Bubble” содержатся файлы, созданные программой Visual Studio.

*Сортировка Хоара*

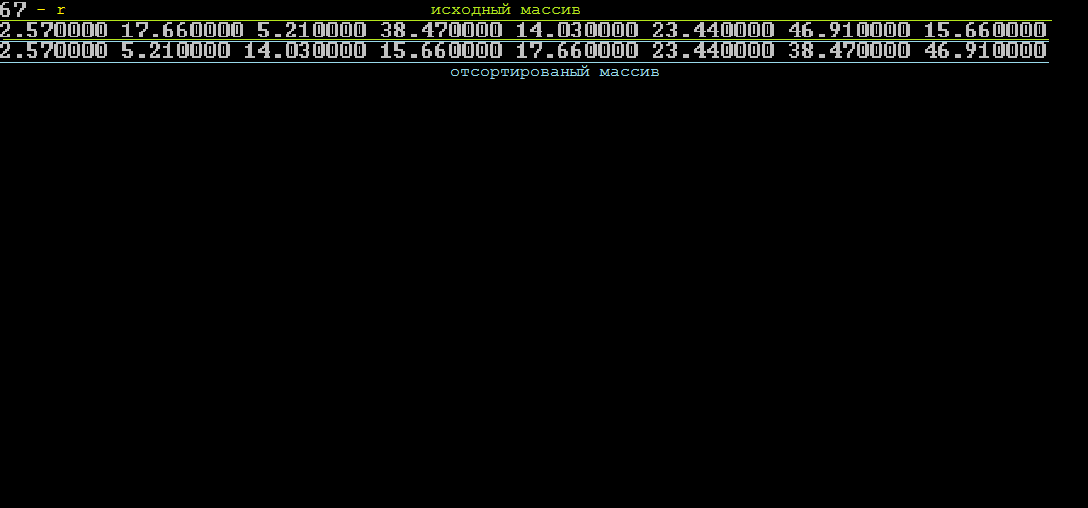
Программа запрашивает целое число от 1 до 100 – кол-во эл. массива. Заполняет массив случайными числами. Выводит исходный массив. Сортирует его. Выводит отсортированный массив. Запрашивает целое число. Выход из программы.



Программа запускается при помощи исполняемого файла “ConsoleApplication5.exe”, при помощи которого запускается сама сортировка (вынесен отдельно для удобства проверки). Так же в папке “ConsoleApplication5” содержатся файлы, созданные программой Visual Studio.

*Сортировка слиянием*

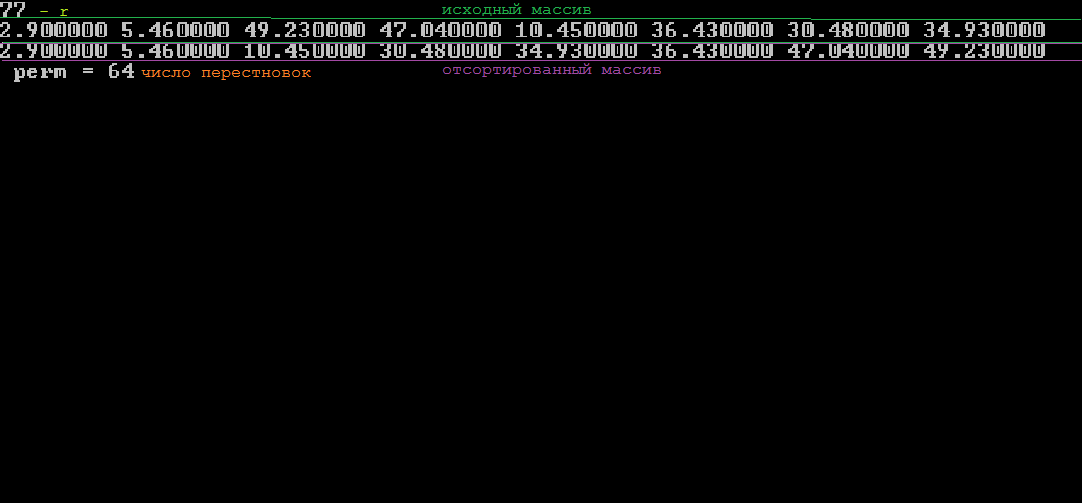
Программа считывает натуральное число (r), на основе которого происходит заполнение массива случайными числами (srand(r)). Формируется массив из 8-ми случайных чисел, который потом выводит на экран. Массив сортируется, отсортированный массив выводится на экран.



Программа запускается при помощи исполняемого файла “mergesort.exe”, при помощи которого запускается сама сортировка (вынесен отдельно для удобства проверки). Так же в папке “mergesort” содержатся файлы, созданные программой Visual Studio.

*Поразрядная сортировка*

Программа считывает натуральное число (r), на основе которого происходит заполнение массива случайными числами (srand(r)). Формируется массив из 8-ми случайных чисел, который потом выводит на экран. Массив сортируется, отсортированный массив выводится на экран.

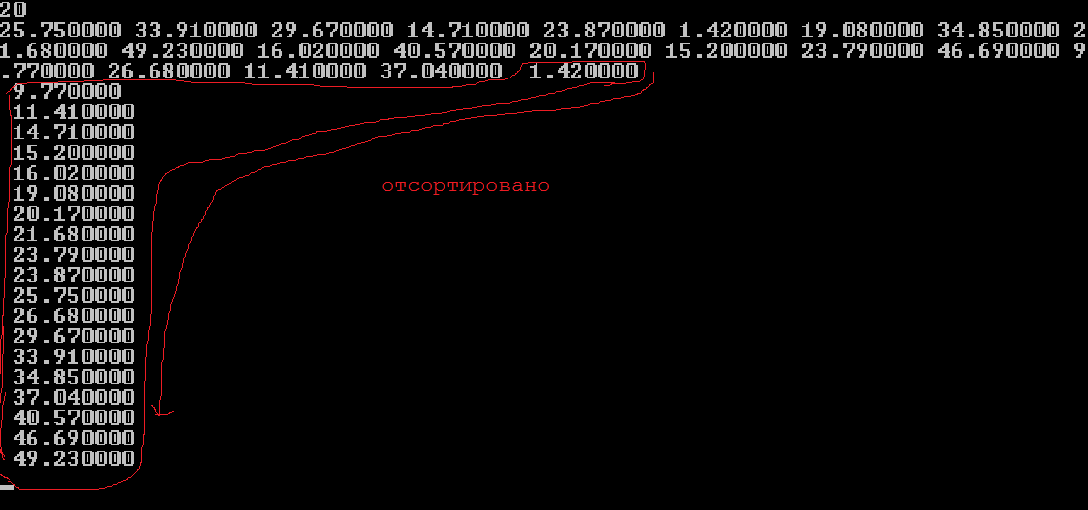
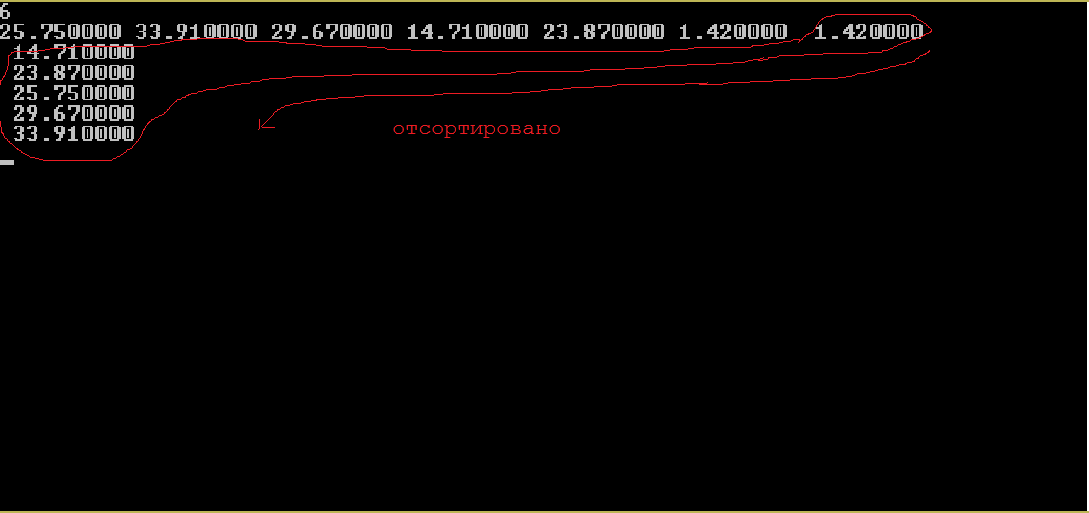
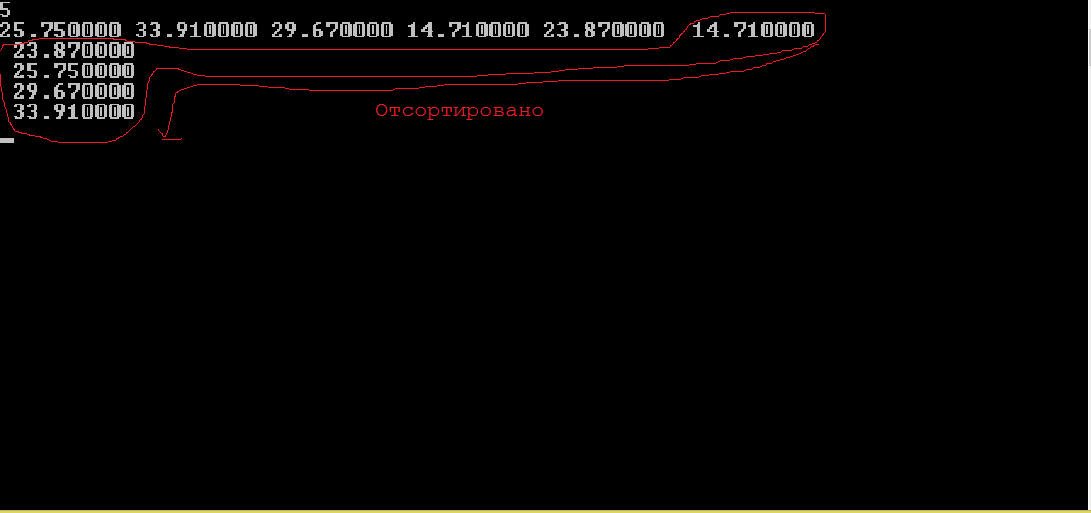


Программа запускается при помощи исполняемого файла “radsort.exe”, при помощи которого запускается сама сортировка (вынесен отдельно для удобства проверки). Так же в папке “radsort” содержатся файлы, созданные программой Visual Studio.

# Подтверждение корректности

*Сортировка пузырьком*

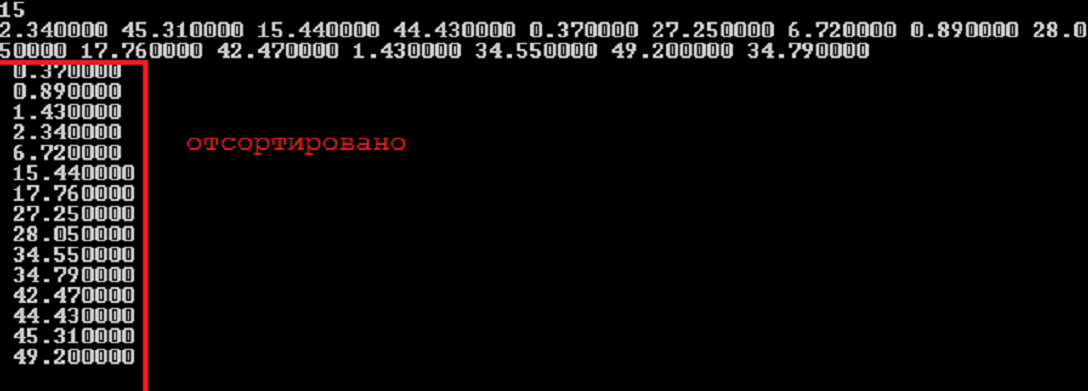
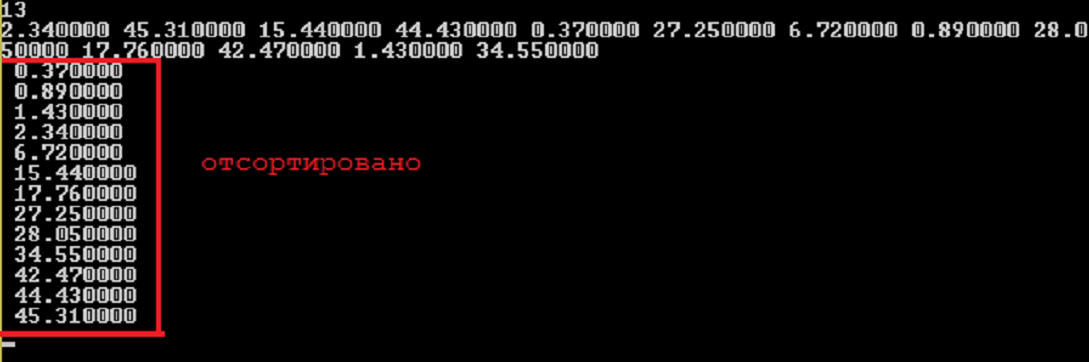
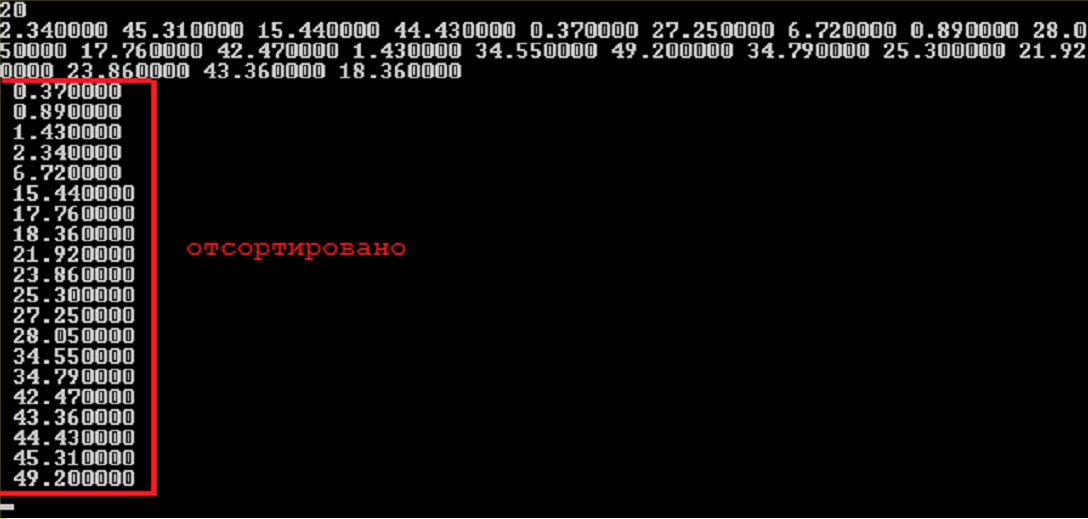
Для подтверждения корректности в программе запустим её с разными входными данными.



Как видим, программа сортирует правильно.

*Сортировка Хоара*

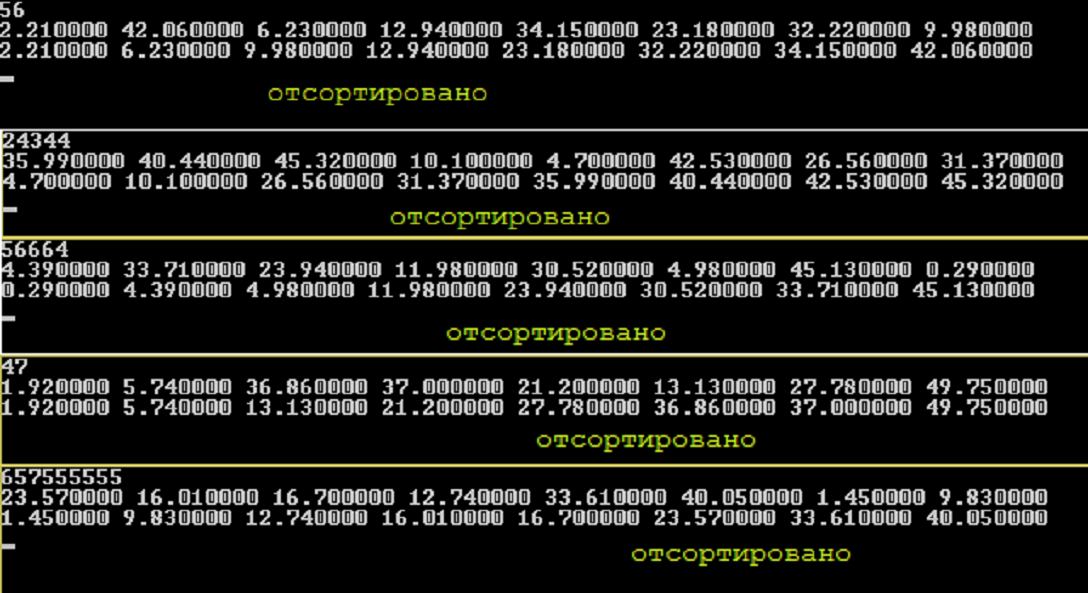
Для подтверждения корректности в программе запустим её с разными входными данными.



Как видим, программа сортирует правильно.

*Сортировка слиянием*

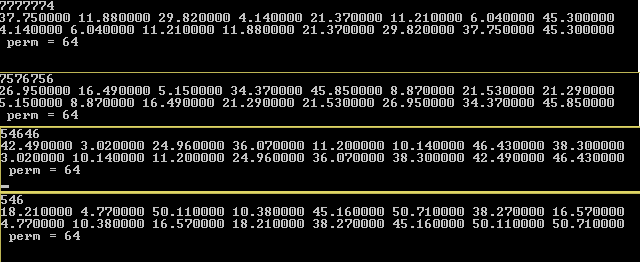
Для подтверждения корректности в программе запустим её с разными входными данными.



Как видим, программа сортирует правильно.

*Поразрядная сортировка*

Для подтверждения корректности в программе запустим её с разными входными данными.

**

отсортировано

отсортировано

отсортировано

отсортировано

Как видим, программа сортирует правильно.

# Результаты экспериментов

*Сортировка пузырьком*

По данным экспериментов видно, что сортировка проходит верно. Сортировка стабильна, отсутствуют «трудные данные» (т.к. реализация максимально проста), не занимает дополнительную память. Однако занимает много времени, что видно при увеличении кол-ва элементов. Очень удобна в сортировке массивов небольшого размера.

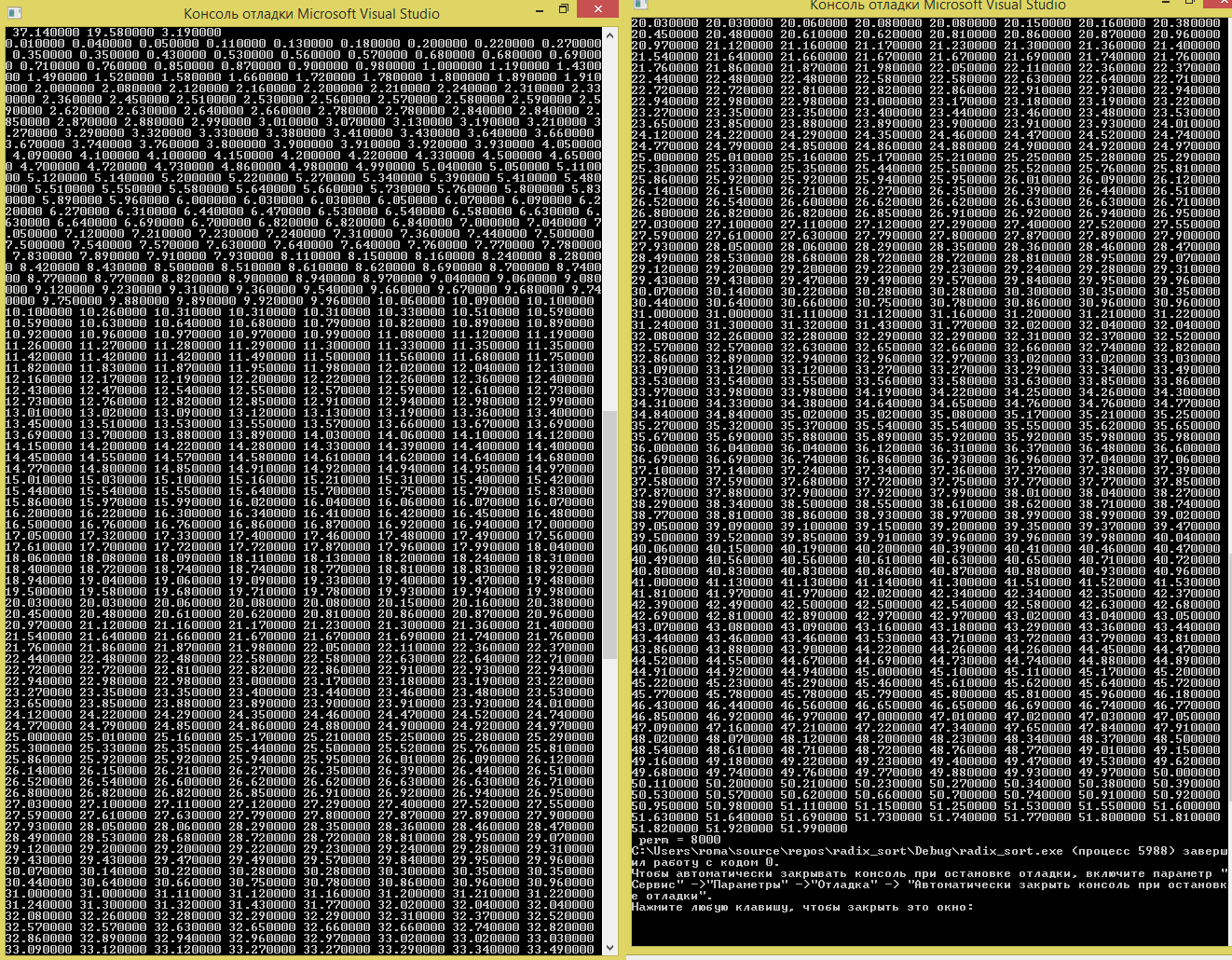
*Сортировка Хоара*

По данным экспериментов видно, что сортировка проходит верно. Программа сортирует довольно быстро. Однако не настолько устойчива. Существуют «трудные» данные, при которых сортировка может деградировать по скорости до О(n2). Кроме того, в плохом случае потребуется сделать до О(n) вложенных рекурсивных вызовов, что может привести к переполнению стека.

*Сортировка слиянием*

По данным экспериментов видно, что сортировка проходит верно. Программа сортирует довольно быстро. Устойчивая. Не имеет «трудных» входных данных. Однако работает примерно одинаково долго как на почти отсортированных массивах, так и на хаотичных, а также требует дополнительную память.

*Поразрядная сортировка*

Сложность этого алгоритма равняется О(width"(n+range)), то есть фактически О(п). При этом требуется дополнительной памяти О(n+width). Из-за сложной реализации и затрат доп. памяти неразумно использовать эту сортировку для массивов малого размера. Зато эта сортировка идеально сортирует массивы огромных размеров. Вот например как программа буквально за 1-2 секунды работы сортирует массив из 1000 элементов:

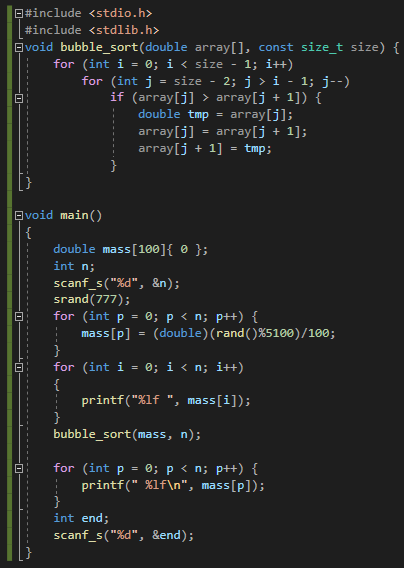
Массив из 100000 элементов программа отсортировала примерно за 10 секунд, при чём большинство времени занял вывод элементов на экран

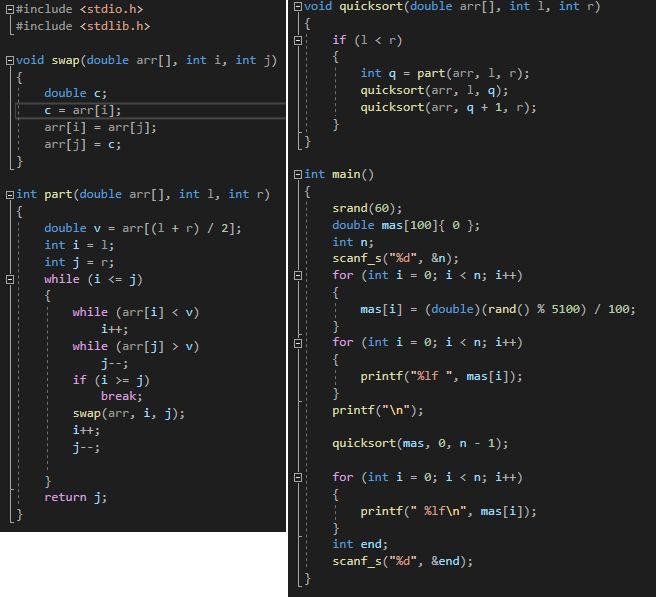
# Заключение

Я изучил алгоритмы сортировки массива: пузырьковой, Хоара, слиянием, поразрядной. Освоил основные приемы использования массивов, методов доступа к элементам массивов, их реорганизации и модификации. В ходе работы выяснилось, что сортировка пузырьком проста в реализации, легко пишется «из головы», стабильна, однако занимает много времени, и скорость работы очень быстро падает с ростом количества эл. в массиве. Сортировка Хоара довольно быстрая, но не очень стабильна, существуют «трудные данные» (например ведущие к переполнению стека или сильной деградации по скорости). Очень удобна в сортировке массивов среднего размера. Сортировка слиянием довольно быстрая, устойчивая, не существуют «трудных данных». Однако требует дополнительную память. Довольно удобна в сортировке массивов почти любого размера, кроме, разве что, совсем огромных. Поразрядная сортировка сложна в реализации, нестабильна, занимает дополнительную память, однако позволяет быстро сортировать массивы огромных размеров.

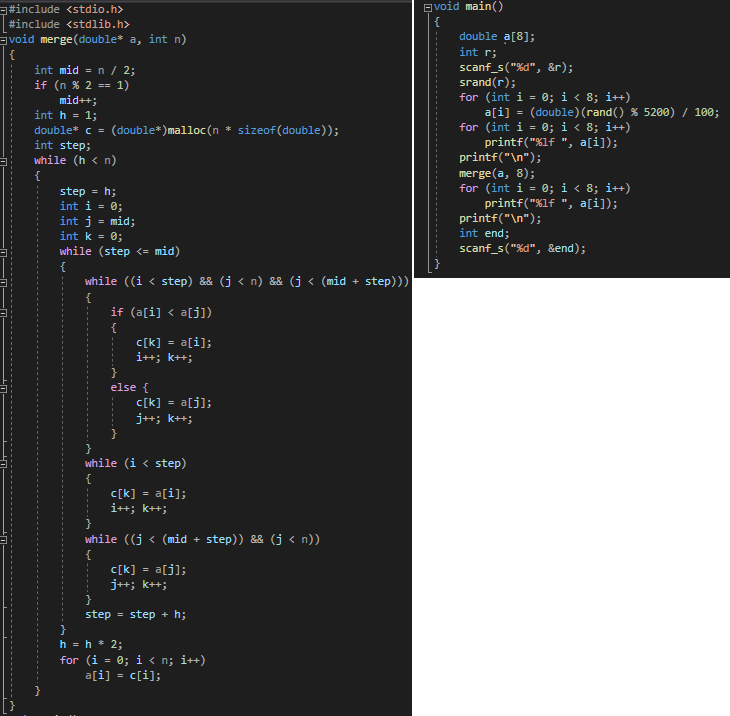
# Приложение

*Сортировка пузырьком*



*Сортировка Хоара*

*Сортировка слиянием*



*Поразрядная сортировка*

