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 массивов»**

**Выполнил**:

студент группы 382003-1

Викулов М.А.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2019

1

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563) - 7

[Руководство пользователя 8](#_Toc26962564)

[Описание программной реализации 9-10](#_Toc26962565)

[Подтверждение корректности 11](#_Toc26962566)

[Результаты экспериментов 12](#_Toc26962567) - 17

[Заключение 18](#_Toc26962568)

2

# Постановка задачи

Цель моей лабораторной работы заключалась в исследовании алгоритмов сортировки массивов, состоящего из элементов типа double (числа с плавающей точкой), а также провести несколько экспериментов с различными входными данными для подтверждения корректности работы того или иного алгоритма и для достоверности эффективности каждого из алгоритмов.

В моей работе были рассмотрены следующие сортировки:

1) Сортировка вставками. Сложность - O(n^2) В средних случаях.

2) Сортировка слиянием. Сложность - О(n logn)

3) Поразрядная сортировка. Сложность - O(m \* (n + k)),

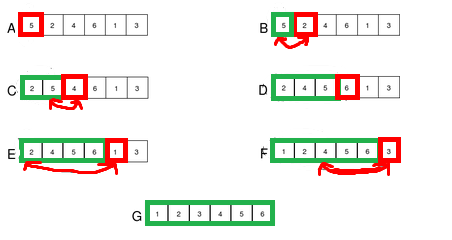
4) Сортировка Шелла, сложность которой рассмотрим далее.

3

# Метод решения

**1) Сортировка вставками.**

Рассмотрим алгоритм данной сортировки на конкретном примере (работать в примере будем с целыми числами, поскольку для чисел с плавающей точкой все корректно работает аналогично).



Идея алгоритма заключается в следующем.

1) Первый элемент массива 5 считается уже вставленным в свое место(соответственно, если в массиве изначально был один элемент, алгоритм корректно завершает свою работу).

2) Берется следующий второй элемент 2. "Вставляем" его в свое место, для чего мы сравниваем его с предыдущими. Если в конце таких сравнений он оказался меньше следующего, но больше предыдущего, то вставлен этот элемент правильно. Важно понимать, что если элемент изначально является наименьшим в массиве, то обязательно необходимо проверить, не выходим ли мы за границу нашего массива. В нашем случае 2 меньше 5, поэтому вставляем его в первую позицию.

3) Следующий элемент на входе 4. По тому же принципу он сравнивается с 2 и 5, и вставляется между ними.

.

.

.

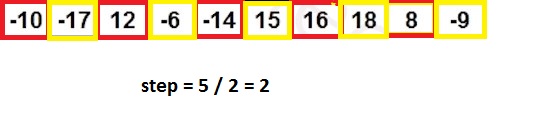
6) Рассматриваем последний элемент массива 3. По тому же принципу вставляем его в свое место между 2 и 4. После этого шага массив отсортирован в порядке неубывания.

4

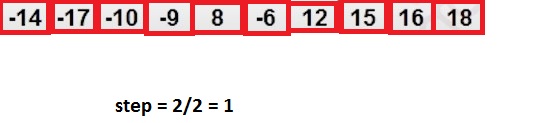
**2)Сортировка Шелла.**

Это алгоритм сортировки вставками с предварительными «грубыми» проходами.

1 шаг



2 шаг



3 шаг



4 шаг

Идея алгоритма: 1) Для этого самого грубого "прохода" мы выбираем расстояние, на котором находятся элементы данного массива (для удобства, возьмем массив целых чисел). Это расстояние обозначено как step. На первой итерации расстояние = =(размер массива)//2{целая часть}. После чего элементы, стоящие на этом расстоянии, начинаются сортироваться алгоритмом сортировки вставками, рассмотренным раннее.

2) Затем расстояние еще делим пополам нацело, тем самым получая step = 5//2 = 2. Соответствующим образом сортируются также сортируются элементы на таком расстоянии.

.

.

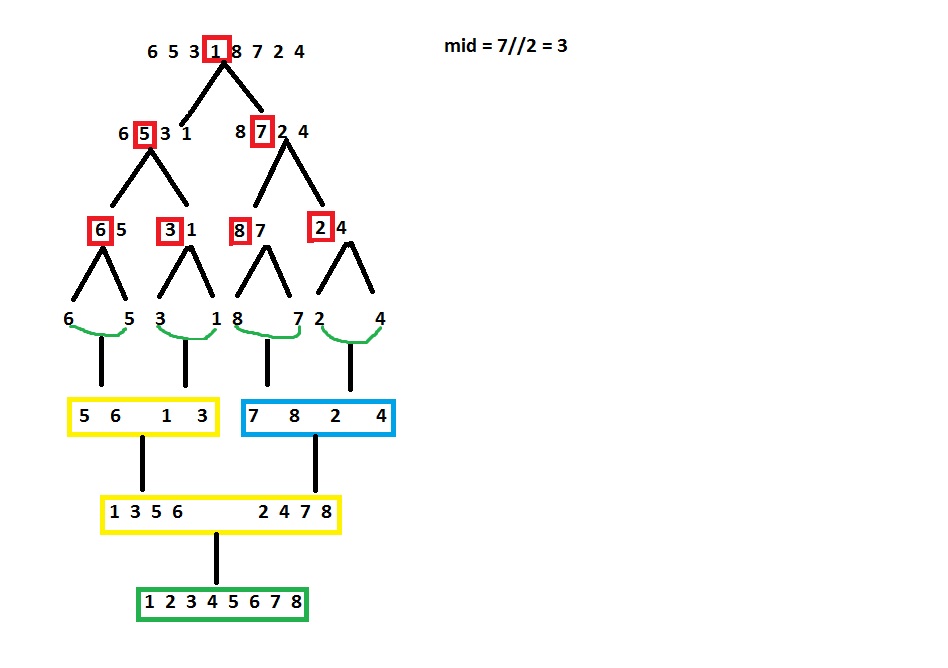
.

3) Если массив еще не отсортирован, то на последнем шаге step = 1. В таком случае, начинает работать обычная сортировка вставками, но уже с меньшим кол-вом перестановок, так как некоторые элементы уже отсортированы. После завершения данный массив будет отсортирован в порядке неубывания.

5

**3)Сортировка слиянием.**

Это рекурсивный алгоритм сортировки, который разделяет данный массив на кусочки, после чего объединяя их в один массив, но уже в порядке неубывания (невозрастания).



1) Выбираем середину массива - это элемент = 1 с индексом 7 // 2 = 3. Делим его на левую и правую часть, тем самым получив левый подмассив с элементами до 1 включительно, и правый подмассив с элементами после 1 и до конца массива.

6

2) Аналогично выполняем до тех пор, пока не разобьем начальный массив на подмассивы, состоящие из одного элемента.

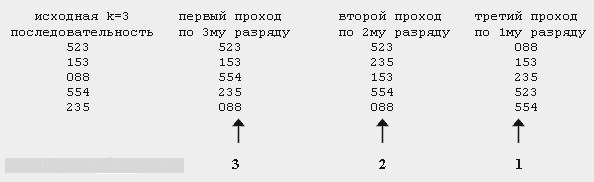
3) Далее начинаем "слияние" этих подмассивов. Начинаем сравнивать соседние элементы и склеивать их, тем самым получив подмассивы размером 2. Далее сравниваем эти подмассивы с соседними, тем самым получив подмассивы размером 4. И так далее будет происходить слияние до тех пор, пока размер "склеенного" не будет равен размеру исходного массива. После чего массив будет отсортирован в порядке неубывания.

6

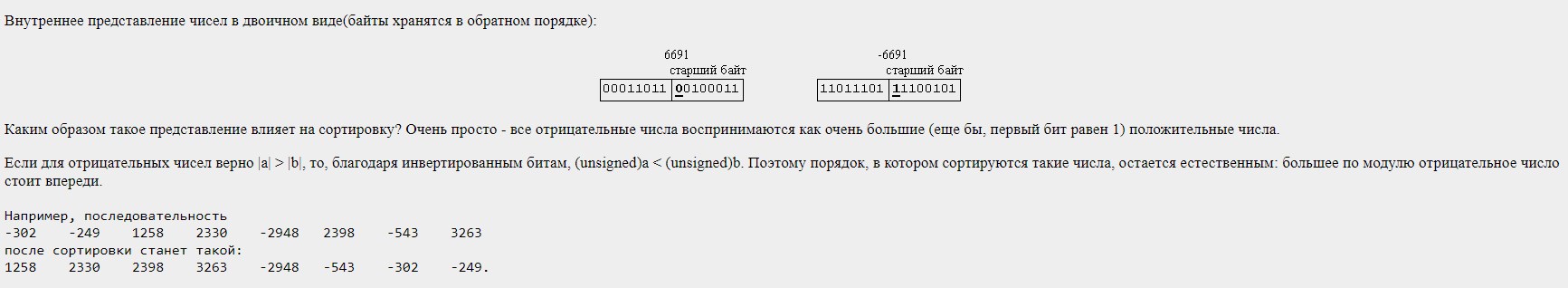
**4) Поразрядная сортировка.**

В этой сортировке предполагается, что каждый ключ сортировки можно рассматривать как K-значное число, каждая цифра которого находится в диапазоне от 0 до m-1. Также в этом алгоритме используется сортировка подсчетом, которая является устойчивой. Используется для каждой цифры, сортируя их, при этом выполняясь справа налево. Для описания данного алгоритма рассмотрим следующую иллюстрацию(здесь для беззнаковых целых чисел).

7



Если рассматривать сортировку со знаковыми числами...

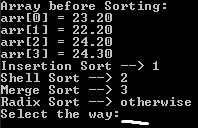


# Руководство пользователя.

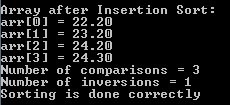
1) Необходимо ввести размер массива (>= 1, иначе программа выведет "ERROR!!!", а также размер должен являться числом). После чего нужно по очереди вводить элементы в любом порядке.



2) Программа предварительно выведет тот массив, который ввел пользователь (неотсортированный), после чего появится меню, в котором пользователю предлагается 4 варианта сортировки. 1 - вставками, 2 - сортировка Шелла, 3 - сортировка слиянием, остальные случаи - поразрядная сортировка.



3) После чего программа выведет в заголовке проделанную сортировку, выведет отсортированный массив на экран и укажет 2 числа, где первое - число сравнений, а второе - число перестановок. Также программа выведет, сортировка прошла корректно или нет.

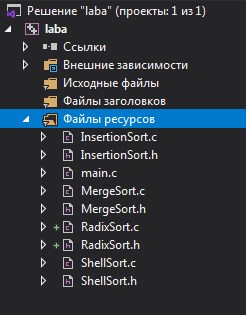


8

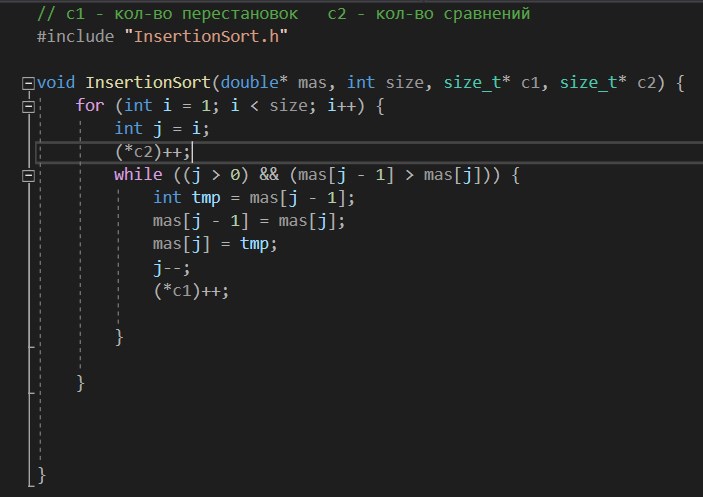
# Описание программной реализации.

Структура проекта, какие есть файлы и что в них содержится.

В моем проекте, не считая файла с функцией main, 4 заголовочных файла и 4 файла с расширением .с. В самом main.c описан создание массива, переменных, ввод, вывод массива, сортировка пузырьком, с помощью которой мы будем проверять корректность того или иного алгоритма, и меню, с помощью которого необходимо выбрать сортировку.

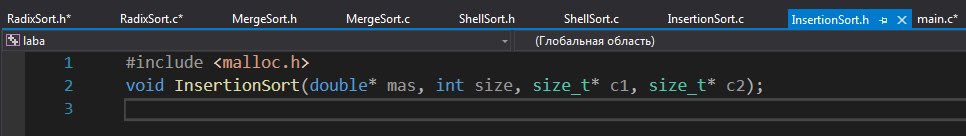


В остальных файлах описываются каждая из сортировок (к примеру, сортировка вставками).



Описание функции сортировки

9



Описание функции сортировки в заголовочном файле

В оставшихся файлах идет аналогичное описание функций других сортировок.

10

# Подтверждение корректности.

Для подтверждения корректности в программе используется сортировка пузырьком массива check - копия данного массива элементов. Для проверки равенства между каждыми элементами массивов создана переменная логического типа flag, которая в случае хотя бы одного неравенства между элементами примет значение "false". Вывод на экран о корректности описан немного выше.

11

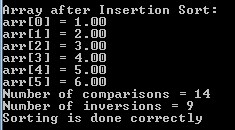
# Результаты экспериментов

По данным экспериментов видно, что …

Проведем несколько экспериментов с входными данными для предварительного выяснения кол-ва сравнений и перестановок, чтобы иметь какое-то представление о сложности того или иного алгоритма.

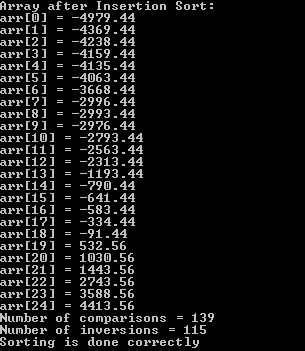
**1) Сортировка вставками.**

- Положим числа в первом эксперименте такие, какие были указаны в описании алгоритма, а именно пос-ть чисел **5 2 4 6 1 3.**

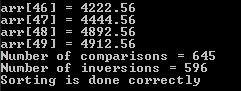
****

Число сравнений при такой пос-ти = 14, а перестановок = 9. Пока на этом шаге сложно говорить об оценке сложности алгоритма.

- Для этого проведем 2 эксперимента: первый со случайными 25 числами в диапазоне от -5000 до 5000, а второй со случайными 50 числами для приблизительного установления эффективности алгоритма.

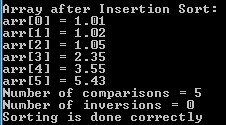
 25 элементов.

12

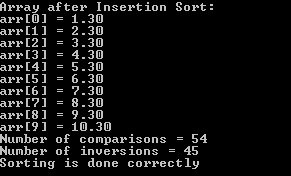
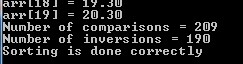
** 50 элементов.

*Из этих экспериментов можно сделать вывод*: при увеличении n(в моей программе size) в 2 раза, кол-во перестановок и сравнений увеличиваются, грубо говоря, в 4 раза. Из этого можно заключить, что сложность алгоритма O(n^2).

Конечно же, лучший случай для данной сортировки это входная в порядке неубывания пос-ть чисел. К примеру, предположим, что наш массив на входе имеет следующий вид: **1.01 1.02 1.05 2.3545 3.5497 5.43223**.



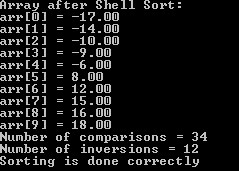
Из этого можно сделать вывод, что в таком случае кол-во сравнений = (размер массива- 1), а кол-во перестановок = 0. Аналогично делаем вывод, что при размере массива = 1000, кол-во сравнений = 999, а кол-во перестановок = 0, т.к. в таком случае все элементы считаются "вставленными" в свое место.

Худший случай для сортировки вставками это упорядоченная пос-ть в обратном порядке (в данном случае, в порядке невозрастания).  

13

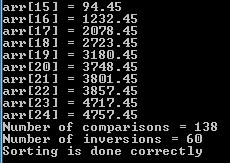
В таком случае, при увеличении размера массива в 2 раза, кол-во перестановок и сравнений возрастает в 4 раза. Сложность остается O(n^2).

**2) Сортировка Шелла.**

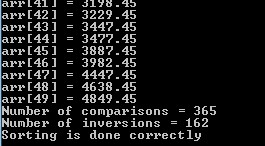
- Положим числа в первом эксперименте такие, какие были указаны в описании алгоритма, а именно пос-ть чисел **15 -17 18 8 -14 -10 16 12 -6 -9 **

После чего проведем также 2 эксперимента со случайными числами:

1) 25 элементов в диапазоне от -5000 до 5000.



2) 50 элементов из этого же диапазона.



В первом эксперименте кол-во сравнений и перестановок = 198, во втором = 527.

Отсюда можем заметить, что при увеличении размера входных данных в 2 раза, кол-во перестановок и сравнений в среднем увеличивается в sqrt((n^3)).

Сложность данного алгоритма зависит от выбора величины step. Есть несколько вариантов пос-тей для оценки сложности, пару из них рассмотрим далее.

14

1) Первоначально используемая Шеллом последовательность длин промежутков:

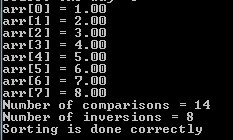
step1 = size/2, step(i) = step(i-1) / 2, step(k) = 1 - где step(k) - худший случай, то есть шаг становится равным единице и начинается обычная сортировка вставками(как в примере описания алгоритма).

2) Предложенная Хиббардом пос-ть чисел 2 4 8 16 32 64... Такая пос-ть чисел приводит к сложности О(sqrt((n^3))).

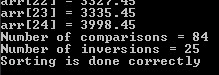
**3) Сортировка слиянием.**

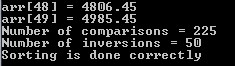
Однозначно говорить про перестановки в данной сортировке нельзя, поскольку это понятие не совсем определено. Если считать слияние подмассивов в единый целый как перестановку эл-тов, то в таком случае кол-во перестановок равно размеру изначального массива.

1)Положим числа в первом эксперименте такие, какие были указаны в описании алгоритма, а именно пос-ть чисел **6 5 3 1 8 7 2 4.**

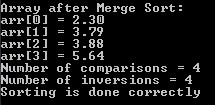
****

2) Далее 2 эксперимента со случайными 25-ю и 50-ю числами в диапазоне от -5000 до 5000.

 25 чисел

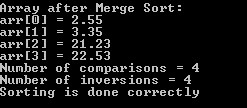
 50 чисел

3) Эксперимент с упорядоченной по неубыванию пос-тью.



15

4) Эксперимент с упорядоченной по невозрастанию пос-тью.



Отсюда можно сделать вывод, что:

|  |  |
| --- | --- |
| **Худшее время** | O(*n* log *n*) |
| **Лучшее время** | O(*n* log *n*) |
| **Среднее время** | O(*n* log *n*) |

+ Для данной сортировки по времени нет таких входных данных, при которых она сильно "деградирует".

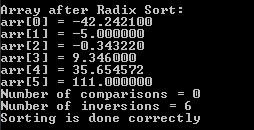
- Большая затрата памяти на вспомогательный массив.

- При отсортированной последовательности время алгоритма O(nlog(n)), в отличие от, например, сортировки вставками O(n).

**4) Поразрядная сортировка.**

Из многих источников мною было установлено, что поразрядная сортировка не использует сравнения при сортировке байтов чисел. Поэтому в моей реализации кол-во сравнений везде = 0. Для приближенной установки эффективности алгоритма проведем несколько экспериментов.

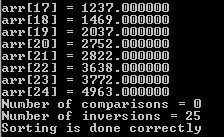
1) Проведем 1 эксперимент с определенной пос-тью чисел для установки корректности алгоритма сортировки. **-0.343220 111 -5 -42.2421 9.346 35.654572**

****

Отсюда можно сделать вывод, что сортировка выполнила свою работу корректно.

16

2) Проведем эксперимент со случайной пос-тью из 25 чисел в диапазоне от -1000 до 1000.



Извините, но кол-во сравнений вывожу с ошибкой... Я никак не смог разобраться, где сравниваются именно элементы массива.

Но все же алгоритм имеет сложность O(m \* (n + k)) по времени, где n - кол-во чисел в массиве, m - диапазон цифр в каждом числе и k - кол-во цифр в числе.

По памяти данный алгоритм имеет сложность O(n + m).

17

# Заключение.

На данном этапе мы рассмотрели 4 алгоритма сортировки массивов. Конечно же, существует большое кол-во дополнительных сортировок, в том числе и быстрая(которая берется в качестве стандартной). Из моей лабораторной работы можно сделать заключение о том, что эффективность каждого из алгоритмов сортировки разная. Зависит она от многих факторов, но основной из них - входная пос-ть элементов. Также необходимо отметить, что в зависимости от входных данных или расположения элементов(объектов) необходимо выбирать тот алгоритм, который работает быстрее и затрачивает меньше памяти.

18

18