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 3823Б1ПМ1-1

Загрядсков М.А.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2023

**Содержание**

[Постановка задачи 3](#_Toc152527359)

[Метод решения 4](#_Toc152527360)

[Руководство пользователя 7](#_Toc152527361)

[Описание программной реализации 8](#_Toc152527362)

[Подтверждение корректности 9](#_Toc152527363)

[Результаты экспериментов 10](#_Toc152527364)

[Заключение 16](#_Toc152527365)

[Приложение 17](#_Toc152527366)

[Источники 18](#_Toc152527367)

# Постановка задачи

Требовалось:

1. Реализовать 4 различных вида сортировок массивов чисел на языке программирования С используя 32-битный тип данных с плавающей запятой (float): сортировка вставками, сортировка Шелла, сортировка слиянием и поразрядная сортировка в сторону менее значащих цифр (LSD);
2. Создать программу с понятным интерфейсом, позволяющим взаимодействовать с реализациями сортировок;
3. Провести тестирование сортировок, проверить их на корректность;
4. Исходный код программ загрузить в указанную репозиторию;
5. Провести ряд экспериментов, в ходе которых доказать, что время исполнения сортировки соответствует теоретической сложности её алгоритма, оценить погрешность, построить графики зависимости времени исполнения, деленного на функцию сложности алгоритма, от количества данных.

# Метод решения

Для решения поставленной задачи, необходимо реализовать все 4 вида сортировок. Приведу описание их алгоритмов, а также алгоритмов некоторых вспомогательных функций:

1. **Сортировка вставками** (Insertion sort). Поделим исходный массив данных на отсортированную левую и неотсортированную правую части. Будем поддерживать отсортированность левой части путем вставки новых элементов из правой, для простоты будем использовать самый ближайший к левой части правый элемент. По умолчанию размер отсортированной части примем за 0, неотсортированной – за n, где n здесь и далее – количество данных в массиве (размер массива). Вставляем по следующему алгоритму: сравниваем вставляемый элемент с текущим элементом левой части, если вставляемый элемент меньше текущего, меняем его местами с текущим элементом и повторяем со следующим элементом из левой части. Выполняем до тех пор, пока вставляемый элемент не будет больше или равен текущему из отсортированной части (см. ист. 1). Очевидно, что если на вход алгоритму подан отсортированный массив, то он за n операций проверит его на отсортированность и завершится. Значит, временная сложность алгоритма в лучшем случае составляет О(n), в худшем и среднем – О(n2) (см. ист. 2).
2. **Сортировка Шелла** (Shell sort). Использует тот же алгоритм, что и сортировка вставками, но позволяет менять местами не только элементы, стоящие рядом, но элементы, находящиеся на определенном расстоянии. Для этого поделим массив на подмассивы элементов, отстоящих друг от друга на шаг S в исходном массиве, а затем отсортируем этот подмассив сортировкой вставками. После, уменьшим шаг S и проделаем то же самое. Понятно, что при шаге S = 1 сортировка Шелла вырождается в обычную сортировку вставками. В каждой из промежуточных стадий сортировки участвуют либо сравнительно короткие массивы, либо уже сравнительно упорядоченные массивы, поэтому алгоритм вставок выполняется сравнительно быстро. Теоретические оценки сложности алгоритма таковы: О(n2) в худшем случае, O(n\*log2(n)) в лучшем и варьируется в зависимости от выбранного алгоритма уменьшения шага S в среднем случае (подробнее см. ист. 3). Из последовательностей, мной были реализованы последовательности Шелла, Хиббарда, Пратта и Мартина Циура для размеров массива от 2000 до 4000 элементов.
3. **Сортировка Слиянием** (Merge sort). Работает по следующему алгоритму: сначала разбиваем массив на n отсортированных массивов длины 1, а затем алгоритмом слияния соединяем их в n/2 отсортированных массивов длины 2, затем в n/4 отсортированных массивов длины 4 и т.д. до тех пор, пока не останется один отсортированный массив длины n. Алгоритм слияния имеет временную сложность О(n) и выполняется так: с первого до последнего элемента каждого массива сравниваем их друг с другом, затем в результирующий массив кладем наименьший из них, а в том массиве, из которого взяли элемент, берем следующий, и так до тех пор, пока элементы хотя бы в одном из массивов не закончатся, после чего перекладываем все элементы из другого массива в конец результирующего (см. ист. 4). Сортировка при любых исходных данных имеет сложность О(n\*lon(n)), что легко доказать: пусть – время сортировки массива длины . Значит, справедливо = где – время, необходимое чтобы слить два массива суммарной длины (см. ист. 5).
4. **Поразрядная сортировка** (Radix sort). Основана на сортировке подсчётом, которая использует ограниченность диапазона данных для сортировки без их сравнения. Принцип таков: создается вспомогательный массив из D+1 нулей, где D – разность между максимальным и минимальным значениями диапазона. Допуская, что исходные данные неотрицательны, проходим по исходному массиву и прибавляем единицу к ячейке вспомогательного массива с индексом, равным значению текущего элемента. После полного прохода по исходному массиву формируем отсортированный массив: проходим по вспомогательному массиву, кладем в результирующий массив столько элементов, равных текущему индексу вспомогательного массива, чему равно значение элемента по данному индексу.  
   Для поразрядной сортировки произвольных чисел используем их байтовое представление в системе. Тогда вспомогательный массив будет иметь размер D = 256 – диапазон значений однобайтового беззнакового целого числа плюс единица. Будем сортировать исходные числа, начиная с наименее значащего байта и заканчивая наиболее значащим, сортировать числа по байтам будем сортировкой подсчётом. После W итераций, где W – объем исходного типа данных в байтах, массив будет отсортирован. После сортировки массива по одному разрядку важно при составлении промежуточного массива сохранять тот порядок, который имели числа до сортировки (см. ист. 6). Сложность такого алгоритма при любых исходных данных, очевидно, составляет O(W\*n). При сортировке отрицательных целых чисел из-за специфики их представления в памяти, после сортировки мы получим сначала все неотрицательные числа в правильном порядке, а затем отрицательные числа в обратном по порядке по их модулю (то есть тоже в правильном). При сортировке чисел с плавающей запятой неотрицательные числа будут отсортированы в правильном порядке (поскольку в их представлении наименее значащие байты находятся с той же стороны, что и у целых чисел), а отрицательные – в правильном порядке по модулю (то есть в обратном). Необходимо учитывать эти особенности при формировании отсортированного массива при работе с указанными типами данных.

# Руководство пользователя

Программа одинаково работает на 64 – битных Windows и Unix системах. Взаимодействие с программой осуществляется через консоль. Последовательность и формат ввода описаны в программе. При некорректном вводе программа выведет “Wrong input” и завершит работу (или просто завершит работу). В ходе взаимодействия с программой можно выбрать тип сортировки, количество данных в массиве, заполнить их случайно или ввести с клавиатуры, для случайного заполнения доступны параметры левой и правой границы, а также ключ генерации. Для сортировки Шелла доступны 4 различных последовательности, по которым будет сортироваться массив. В конце можно вывести отсортированный массив на экран.

# Описание программной реализации

В папке проекта доступны следующие файлы:

1. Заголовочный файл с описанием всех используемых в программе функций SortLab.h.
2. Файл реализации этих функций SortLab.c.
3. Файл решения Visual Studio SortLab.sln.
4. Файл проекта Visual Studio SortLab.vcxproj.
5. Файл Test.c, проверяющий сортировки на корректность.
6. Файл TimeMeasure.c, для подсчёта времени выполнения сортировок и загрузки их в файл (работает только на Unix – системах в связи со спецификой измерения процессорного времени).
7. Файл UserProgram.c – код программы, позволяющей взаимодействовать с пользователем через консоль.
8. Файл uniexe – исполняемый файл для Unix – систем. Исполняемый файл для Windows не был добавлен, т.к. расширение .exe добавлено в исключения в файле .gitignore.

# Подтверждение корректности

Для подтверждения корректности в программе были реализованы следующие функции:

* Функция удаления заданного элемента из отсортированного массива. Средняя сложность – S1 = О(n). Возвращает -1, если такого элемента не нашлось.
* Функция проверки массива на отсортированность, которая использует изначальный и отсортированный наборы данных, после чего последовательно удаляет из отсортированного массива элементы из исходного. Если подтвердилось, что в проверяемом массиве элементы отсортированы, а после удаления всех элементов проверяемый массив имеет нулевую длину – проверяемый массив отсортирован корректно. Средняя сложность – S2 = О(S1) = O(n2).

Недостатки такого способа очевидны: функция проверки работает слишком долго, а также требует скопировать элементы из исходного массива в сортируемый, чтобы продолжить с ним работу дальше. Я использовал именно этот способ, чтобы на ранних стадиях разработки без применения сторонних библиотек следить за корректностью своих алгоритмов, а поскольку данный алгоритм проверки мне кажется значительно проще, чем некоторые алгоритмы сортировок, такую функцию легко отладить, а затем определить некорректные алгоритмы сортировок.

В ходе тестирования своих алгоритмов сортировок, я 1000 раз сортировал массив, размера 1000, каждой сортировкой отдельно, с разными ключами генерации случайных чисел после чего проверял на отсортированность. Если какая-либо сортировка не прошла проверку, то выводился ключ и та сортировка, которая её не прошла. При прохождении всех сортировок корректно программа завершает работу.

# Результаты экспериментов

Эксперименты проводились на устройствах с установленными на них Unix – системах (MacOS 13.4.1, процессор Apple M1, дальше просто М1; GNU/Linux 11, версия ядра 6.1.21-v8+, процессор Broadcom BCM2711, дальше просто Rasp).

Сначала я решил оценить среднюю теоретическую временную сложность. Для её подтверждения было проведено 9000 измерений каждой сортировки на каждом из устройств с ключом генерации случайных чисел равным 5 на размерах массивов от 1000 до 9999 включительно. Замеры выполнялись последовательно для каждой сортировки для определенного значения размера. Для сортировки Шелла использовалась последовательность Хиббарда, которая работала быстрее всего на данных размерах массивов по моим наблюдениям в ходе экспериментов. Затем было предположено, что временная сложность алгоритмов верна, а значит, при делении экспериментального времени на функцию внутри О-большое будет получена константа, небольшой разброс значений которой будет доказывать временную сложность. Изначально были выбраны следующие функции: для сортировки вставками: , для сортировки Шелла: , для сортировки слиянием: , для поразрядной сортировки: , где n – размер массива. После нахождения для каждой из сортировок, где – время, поделенное на соответствующую функцию временной сложности, , стало понятно, что первые 58 итераций не соответствовали ожидаемому времени сортировки: в них время быстро уменьшалось с увеличением размера массива, поэтому было принято решение начать анализ данных с размера . Также сразу стало понятно, что разброс значения слишком велик, поэтому для сортировки Шелла была введена другая функция, соответствующая лучшему случаю сортировки: .

После вычисления всех значений были подсчитаны их средние значения, а затем среднеквадратичные отклонения: , число попавших в зону нормального распределения элементов определим как %. Чтобы сравнивать различные оценки алгоритмов, я введу понятие относительного среднеквадратичного отклонения, которое будет равно , где – среднее значение времени в выборке: Из данных значений можно сделать вывод, что все выборки соответствуют нормальному распределению. Функция оказалась в 5 раз точнее по сравнению с функцией . Графики зависимости от привожу ниже:

Здесь представлены сортировки, имеющие значение сложности между O(n2) и О(n).

Здесь все остальные.

Как можно заметить в том числе из графиков, под свою оценку лучше всего подходят функции и . Практически все линии на диаграмме имеют тенденцию «убывания», что связано с самой концепцией оценки О-большое: пусть действительная оценка временной сложности алгоритма равна Тогда , а На диаграмме это выглядит как «убывание».

Можно заметить и другую особенность, что разброс значений на графике тем больше, чем меньше среднее время выполнения программы. Это связано с относительной погрешностью измерения процессорного времени, и это мешает точно доказать временную сложность на практике.

Так как же доказать временную сложность? Самая медленнорастущая при достаточно больших функция, используемая в оценке сложности – . Среди элементарных функций самая медленнорастущая, из мне известных, это конечная композиция логарифмов где количество композиций – конечное число. Я решил взять функцию которая является а затем разделить или умножить исходные на неё, после чего вычислить относительное среднеквадратичное отклонение. Пусть соответствующее значение для , а для Приведу соответствующие значения ниже:

,

,

.

Видно, что теоретически предположенные функции оказались наилучшими на практике в алгоритмах 1 и 4. В то же время, полученные функции оказались лучшими по практической оценке для алгоритмов 3 и 5, исходя из их относительного среднеквадратичного отклонения. С логической точки зрения это, например, для поразрядной сортировки, означает, что временная сложность алгоритма поразрядной сортировки функция, более медленнорастущая, чем O(n). Но такое невозможно, так как для массива с неизвестными исходными данными необходимо проанализировать каждый элемент среди n элементов в массиве хотя бы 1 раз.

Аналогичное исследование проведем и для другого устройства. Ниже приведены значения для тех же алгоритмов, протестированных на устройстве Rasp:

Графики зависимости :

Сортировки, имеющие значение сложности между O(n2) и О(n).

Остальные сортировки.

На графике сортировки слиянием на значении происходит резкий рост, хотя на других графиках такого не наблюдается. Я связываю это с превышением размера кэша L1 процессора в 2\*4096\*4 = 32Кб, где 2 – количество массивов (вспомогательный и сортируемый для данной сортировки), 4096 – значение размера массива n, где происходит быстрый рост, 4(байт) – размер типа данных float. На графике поразрядной сортировки присутствует заметная зона убывания примерно до размера в 4000 элементов. Такое поведение диаграммы я объяснить, не могу.

* Среднеквадратичные отклонения: .
* Число элементов, попавших в зону нормального распределения:
* Относительное среднеквадратичное отклонение во всех трёх случаях, указанных выше:

,

,

.

Видно, что многие относительные среднеквадратичные отклонения равны таковым при анализе прошлой выборки, несмотря на совершенно другой диапазон значений выборки и значения среднеквадратичных отклонений. Это может свидетельствовать о том, что такая оценка является довольно точной и независимой, хотя необходимы дальнейшие исследования. Также полученные данные подтвердили теоретическую сложность для всех исследуемых алгоритмов, кроме сортировки Шелла.

Оценка наихудшего случая аналогична среднему случаю для всех сортировок, как и оценка наилучшего случая для сортировки слиянием и поразрядной сортировки, поэтому я не привожу их здесь. Решено было провести отдельное исследование лучшего случая для сортировок вставками и Шелла, но так как время выполнения этих сортировок в этом случае мало, решено было увеличить диапазон размеров массивов с 9000 до 90000. Теоретические оценки лучшего случая для сортировки вставками: , для сортировки Шелла: .

По результатам исследования я получил следующие значения:

* Среднеквадратичные отклонения: .
* Число элементов, попавших в зону нормального распределения:
* Относительные среднеквадратичные отклонения:

Для сортировки вставками значение получилось больше, чем в среднем случае, хотя судя по диаграмме, значение не убывает и не возрастает, а просто обладает большим разбросом (напомню, что диапазон увеличен в 10 раз, значит и убывание с возрастанием на таких масштабах было бы заметнее). Разброс уменьшается с увеличением времени, следовательно, для повышения точности необходимо увеличить диапазон еще больше и поднять верхнюю границу размера массива.

По полученным результатам можно сделать вывод, что с такой точностью измерения времени и такой выборкой строго доказать теоретическую сложность алгоритмов на практике невозможно, но с некоторыми допущениями можно сделать следующие суждения о сложности сортировок: сложность сортировки вставками в среднем случае составила ), сложность сортировки Шелла в среднем случае на практике соответствует ), средняя сложность сортировки слиянием: , средняя сложность поразрядной сортировки: . Сложность сортировки вставками в лучшем случае составила ), сложность сортировки Шелла в лучшем случае на практике соответствует ).

# Заключение

По результатам проведенных экспериментов были сделаны следующие выводы:

* Были реализованы и проверены различные типы алгоритмов сортировок, создана программа для взаимодействия с пользователем под разные типы операционных систем, без графического интерфейса.
* По результатам экспериментов под свою теоретическую оценку сложности лучше всего подходят сортировки вставками и слиянием. Для поразрядной сортировки её теоретическая временная сложность была подтверждена только в одном исследовании, а сортировка Шелла не соответствовала своей теоретической оценке ни в одном случае. Полученные практические оценки сложности привожу ниже:  
  сложность сортировки вставками в среднем случае составила ), сложность сортировки Шелла в среднем случае на практике соответствует ), сложность сортировки слиянием: , сложность поразрядной сортировки: . Сложность сортировки вставками в лучшем случае составила ), сложность сортировки Шелла в лучшем случае на практике соответствует ).
* Из диаграмм и вычисленных отклонений можно заметить, что разброс данных около константы довольно велик, что свидетельствует о недостаточной точности измерения времени и о слишком маленькой выборке.
* Некоторые явления, заметные на диаграммах, не были объяснены.

**Вывод:** для доказательства теоретической временной сложности алгоритмов сортировок необходимо задействовать ресурсы из других языков программирования, кроме С (таких как ассемблер, С++) и использовать более точные измерения процессорного времени, а также провести большее количество исследований на большем количестве устройств. На данном этапе точно доказать сложность алгоритмов невозможно.

# Приложение

Смотрите код в репозитории.

# Источники

1. Д. Э. Кнут, Искусство программирования, 2-е издание, ред. Ю. В. Козаченко,  [ISBN 5-8459-0082-1](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/5845900821) – стр. 99.
2. <https://ru.wikipedia.org/wiki/Сортировка_вставками>.
3. Д. Э. Кнут, Искусство программирования, 2-е издание, ред. Ю. В. Козаченко,  [ISBN 5-8459-0082-1](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/5845900821) – стр. 102 – 115.
4. Д. Э. Кнут, Искусство программирования, 2-е издание, ред. Ю. В. Козаченко,  [ISBN 5-8459-0082-1](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/5845900821) – стр. 181.
5. <https://neerc.ifmo.ru/wiki/index.php?title=Сортировка_слиянием>
6. Д. Э. Кнут, Искусство программирования том 3, 2-е издание, ред. Ю. В. Козаченко,  [ISBN 5-8459-0082-1](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/5845900821) – стр. 192 - 201.