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»**

**Выполнила**:

студентка группы 3821Б1ПМ2

Миронова А. А.

**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 9](#_Toc26962566)

[Результаты экспериментов 10](#_Toc26962567)

[Заключение 12](#_Toc26962568)

# Постановка задачи

Целью лабораторной работы являлась реализовать на языке программирования Си сортировку пузырьком, Шелла, слиянием и поразрядную сортировки. Сортировки нужно реализовать для данных типа double. После работы программы массив должен быть отсортирован от меньшего элемента к большему. Нужно описать программную реализацию и алгоритмы работы данных сортировок. Необходимо подтвердить корректность реализации данных сортировок. Провести эксперименты для подтверждения сложности, описать способ проведения экспериментов и сделать вывод по полученным результатам.

# Метод решения

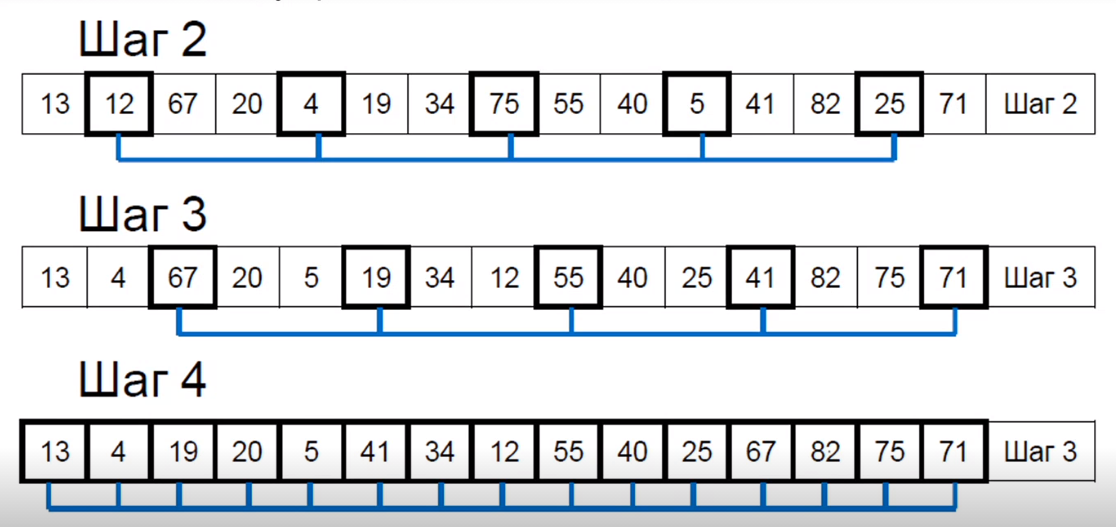
**Сортировка Пузырьком** **(Bubble Sort)**

Алгоритм сортировки «пузырьком» следующий: сортировка массива осуществляется путем последовательного сравнения и обмена соседних элементов, если предшествующий оказывается больше последующего. Проходы по массиву повторяются n-1 (n - кол-во элементов массива) раз. В процессе выполнения данного алгоритма, элементы с большими значениями оказываются в конце массива, очередной наибольший элемент массива ставится рядом с предыдущим наибольшим элементом, а элементы с меньшими значениями постепенно перемещаются по направлению к началу массива. Сложность данного алгоритма сортировки равна O(n2).

**Сортировка Шелла (Shell Sort)**

Сортировка Шелла основана на сортировке вставками. Идея сортировки методом Шелла состоит в том, чтобы сортировать элементы стоящие друг от друга на некотором расстоянии, которое обычно называют step (шаг). Затем алгоритм сортировки повторяется для некоторых меньших значений step, и в конце процесс сортировки Шелла завершается упорядочиванием элементов при step = 1 (а именно обычной сортировкой вставками). В худшем случае сложность алгоритма составляет O(n2) в худшем случае и O(n\*log2(n)) в лучшем случае.





**Сортировка слиянием (Merge Sort)**

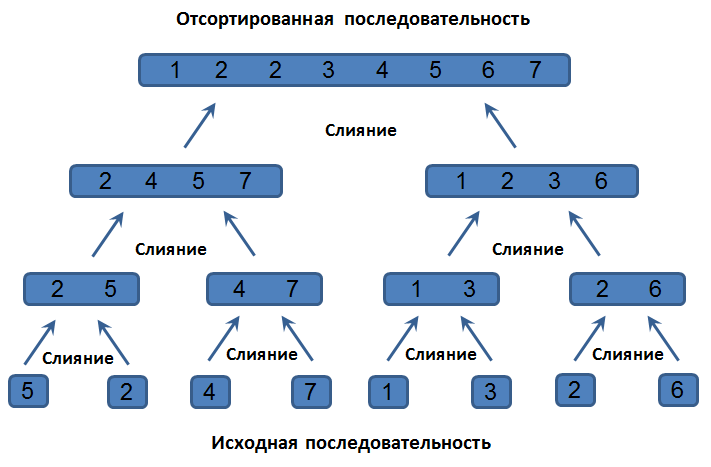
Алгоритм сортировки слиянием основан на идее объединения двух отсортированных массивов. Его можно разделить на три этапа.

Суть первого шага состоит в том, что сортируемый массив рекурсивно разбивается на две части примерно одинакового размера до тех пор, пока размер его размер не будет равным единице.

Затем наступает второй шаг: разделив массив на две части, нужно их отсортировать, нужно каждую часть поделить на две части и отсортировать и т.д. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы, так как любой массив длины 1 можно считать упорядоченным.

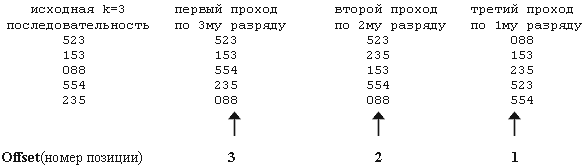
Третьим шагом является слияние двух упорядоченных массивов. Сначала соединяются массивы длины 1 в массивы длины 2, затем массивы длины 2 в массивы длины 4 и т.д. В общем случае слияние подмассивов происходит так: на каждом шаге мы берём меньший из двух первых элементов подмассивов и ставим его в начало результирующего массива. Счётчики номеров элементов результирующего массив и подмассив, из которого был взят элемент, увеличиваем на 1. Когда один из подмассивов заканчивается, добавляются все оставшиеся элементы второго подмассива в результирующий массив.

Сложность алгоритма составляет O(n\*log2(n)) в любом случае, а также необходимо О(n) памяти.



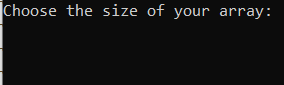
**Поразрядная сортировка (Radix Sort)**

В поразрядной сортировке линейный алгоритм, не использующий сравнения. Изначально сравниваются значения крайнего разряда, в результате этого элементы группируются по результатам этого сравнения. Затем сравниваются значения разряда, стоящего левее предыдущего, после чего элементы либо упорядочиваются по результатам значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются совсем, при этом сохраняя порядок, достигнутый при предыдущей сортировке. Аналогично делается для следующего разряда, и так до конца сортировки. Поскольку сортировка подсчетом является устойчивой, то цифры располагаются в правильном порядке. Существует два варианта этой сортировки: LSD (сортировка с выравниванием по младшему разряду, направо, к единицам) и MSD (сортировка с выравниванием по старшему разряду).

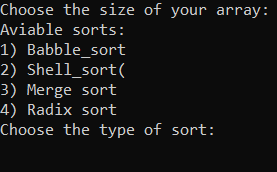


# Руководство пользователя

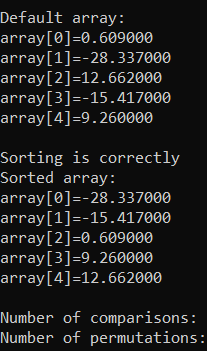
1. При запуске программы пользователь должен ввести размер массива:



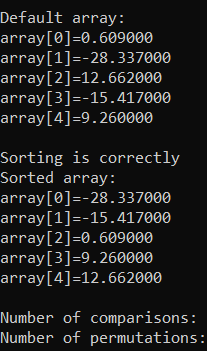
1. Затем показываются возможные способы сортировки (Пузырьком, Шелла, Слиянием и Поразрядной). Нужно ввести номер выбранной сортировки из предложенных по списку:



1. На экране появится случайный исходный массив, оповещение о корректности сортировки и отсортированная версия массива.



1. В конце программа выводит количество перестановок и сравнений. После этого консоль может быть закрыта нажатием любой клавиши.



# Описание программной реализации

**Используемые функции:**

1. void babble\_sort(double\* array, int size)– сортировка Пузырьком, сортирует массив по вышеописанному алгоритму. Принимает указатель на массив типа double и его размер типа int. Ничего не возвращает.
2. void shell\_sort(double\* array, int size)– сортировка Шелла, сортирует массив по вышеописанному алгоритму. Принимает указатель на массив типа double и его размер типа int. Ничего не возвращает.
3. void merge(double\* array, double\* secondhand, int left, int middle, int right) – упорядочивает две половины массива, после чего сортирует их как один целый массив. Принимает указатель на исходный массив и дополнительный пустой массив типа double, номер первого элемента левой части массива, номер последнего элемента левой части массива и номер последнего элемента правой части массива типа int. Ничего не возвращает.
4. void merge\_sort(double\* array, double\* secondhand, int left, int right) – сортировка слиянием. Принимает указатель на исходный и дополнительный массив типа double, номер первого и последнего элемента части массива, которую нужно отсортировать, типа int. Ничего не возвращает.
5. void createCounters(double\* array, long\* counters, int N) – создаёт корзины для всех возможных значений разряда числа, принимает указатель на исходный массив типа double, указатель на выходной массив с корзинами типа long и размер исходного массива. Ничего не возвращает.
6. void radixPass(short Offset, int size, double\* array, double\* dest, long\* count) – записывает элементы массива, отсортированные по заданному разряду. Принимает номер байта типа short, размер исходного массива, указатель на исходный массив типа double, указатель на массив для записи отсортированных чисел типа double, указатель на массив корзин по разряду типа long. Ничего не возвращает.
7. void radixLastPass(short Offset, int size, double\* array, double\* dest, long\* count) – проход по знаковому разряду чисел, перестановка отрицательных чисел в начало массива. Принимает номер байта типа short, размер исходного массива, указатель на исходный массив типа double, указатель на массив для записи отсортированных чисел типа double, указатель на массив корзин по разряду типа long. Ничего не возвращает.
8. void radixSort(double\* array, int size) – поразрядная сортировка. Принимает на вход массив double и его размер. Ничего не возвращает.

# Подтверждение корректности

Для подтверждения корректности в программе реализована функция

void sorting\_check(double\* array, long size), принимающая указатель на уже отсортированный массив, и его размер. В функции сравнивается предыдущий и следующий элемент массива, начиная со второго и заканчивая последним. В случае хотя бы одного элемента, стоящего не по порядку, выводится «Sorting is not correctly». В обратном случае на экран выводится «Sorting is correctly».

# Результаты экспериментов[[1]](#footnote-1)

1. **Сортировка пузырьком**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 100 | | | 200 | | | 400 | | | 600 | | | 1000 | | |
| Х | С | Л | Х | С | Л | Х | С | Л | Х | С | Л | Х | С | Л |
| Сравнения | 4950 | 4929 | 99 | 19900 | 19899 | 199 | 79800 | 79734 | 399 | 179700 | 178619 | 599 | 499500 | 497904 | 999 |
| Перестановки | 4892 | 2371 | 0 | 19793 | 9779 | 0 | 79600 | 38475 | 0 | 179402 | 89873 | 0 | 499003 | 236031 | 0 |

**Сложность алгоритма**:

O(n) в лучшем случае.

O(n^2) в среднем.

O(n^2) худшем случае.

Видно, что число сравнений растет линейно в лучшем случае. В худшем и среднем случаях сложность сравнений и перестановок растет квадратично.

1. **Сортировка Шелла**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 100 | | | 200 | | | 400 | | | 600 | | | 1000 | | |
| Х | С | Л | Х | С | Л | Х | С | Л | Х | С | Л | Х | С | Л |
| Сравнения | ... | 899 | 503 | ... | 950 | 1203 | ... | 5494 | 2803 | ... | 8759 | 4804 | ... | 16258 | 8006 |
| Перестановки | ... | 354 | 43 | ... | 989 | 44 | ... | 2608 | 45 | ... | 3668 | 46 | ... | 8022 | 46 |

**Сложность алгоритма**:

O(n) в лучшем случае.

O(n (log n)²) в среднем случае.

Сложность худшего случая равна сложности в среднем случае. Она зависит не от исходных данных, а от выбора шага step, и потому не рассматривается.

По таблице видно, что в лучшем случае сложность меняется линейно, так как число перестановок постоянно.

1. **Сортировка слиянием**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 100 | | | 200 | | | 400 | | | | 600 | | | | 1000 | | | |
| Х | С | Л | Х | С | Л | Х | С | Л | Х | | С | Л | Х | | С | Л |
| Сравнения | ... | 300 | 356 | ... | 651 | 812 | ... | 1505 | 1824 | ... | | 2419 | 2916 | ... | | 4381 | 5044 |
| Перестановки | ... | 672 | 672 | ... | 1544 | 1544 | ... | 3488 | 3488 | ... | | 5576 | 5576 | ... | | 9976 | 9976 |

**Сложность алгоритма**:

O(n\*log(n)) в лучшем случае.

O(n\*log(n)) в среднем случае .

Сложность в худшем случае здесь так же равна сложности в среднем, и этот случай так же не просто получить, поэтому он не рассматривается.

По таблице хорошо видно, что сложность и в лучшем и худшем случае меняется одинаково линейно-логарифмически.

1. **Сортировка слиянием**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 100 | | | 200 | | | 400 | | | | 600 | | | | 1000 | | | |
| Х | С | Л | Х | С | Л | Х | С | Л | Х | | С | Л | Х | | С | Л |
| Перестановки | ... | 402 | ... | ... | 1600 | ... | ... | 3200 | ... | ... | | 4800 | ... | ... | | 8000 | ... |

**Сложность алгоритма**:

Сложность поразрядной сортировки всегда линейно зависит от n числа элементов и равна O(n\*(m+k)), где m и k постоянные для каждого отдельного случая. Значит достаточно рассмотреть, например, средний случай. Легко убедится, что сложность линейная.[[2]](#footnote-2)

# Заключение

В ходе лабораторной работы на языке программирования Си были реализованы сортировка пузырьком, сортировка Шелла, сортировка слиянием и поразрядная сортировка. Были описаны алгоритмы работы данных сортировок, их программная реализация и проведенные эксперименты для замера и подтверждения их теоретический сложности. В ходе проведения экспериментов была проведена проверка корректности сортировок на большом объеме данных и подтверждена их теоретическая сложность.

1. Х – Худший случай (Обратно отсортированный массив)  
   С – Средний случай (Случайный массив)  
   Л – Лучший случай (Отсортированный массив) [↑](#footnote-ref-1)
2. Х – Худший случай (Обратно отсортированный массив)  
   С – Средний случай (Случайный массив)  
   Л – Лучший случай (Отсортированный массив) [↑](#footnote-ref-2)