Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский Нижегородский государственный университет

им. Н.И. Лобачевского» (ННГУ)

Институт Информационных технологий, математики и механики

Отчёт по лабораторной работе:

«Сравнение алгоритмов сортировки»

**Выполнил:**

студент института ИТММ

гр. 3823Б1ПМ2

Баринов Артём Сергеевич

**Проверил:**

ст. преп. Кафедры ВВиСП   
института ИТММ

Пирова Анна Юрьевна

Нижний Новгород

2024 г.

Содержание

1. [Введение 3](#_Toc270962758)
2. [Постановка задачи 3](#_Toc270962759)
3. [Руководство пользователя 5](#_Toc270962760)
4. [Руководство программиста 7](#_Toc270962761)
   1. [Описание структур данных 7](#_Toc270962762)
   2. [Описание алгоритмов 7](#_Toc270962763)
   3. [Описание структуры программы 7](#_Toc270962764)
5. Вычислительные эксперименты 9
6. [Заключение 9](#_Toc270962765)
7. [Литература 10](#_Toc270962766)
8. [Приложения 11](#_Toc270962767)
9. [Приложение 1 11](#_Toc270962768)
10. [Приложение 2 11](#_Toc270962768)

# Введение

Реализации методов сортировки являются крайне полезными алгоритмами. Все они позволяют достаточно быстро сортировать массивы данных. Однако с увеличением размера сортируемых данных скорость выполнения начинает сильно зависеть от выбранного метода.

В одних случаях сортировка выполняется крайне быстро, а в других слишком медленно. В связи с этим очень актуальным является вопрос выбора подходящего алгоритма сортировки в конкретной задаче.

Целью моей работы является сопоставление по скорости выполнения алгоритмов сортировки данных в массивах разной величины.

Будут рассмотрены наиболее популярные алгоритмы сортировки: методом вставок, метод Шелла, методом пузырька, методом слияния, быстрая сортировка.

Языком реализации был выбран язык Си. Это язык программирования общего назначения, известный своей эффективностью и возможностью интеграции в практически любую среду разработки. Использование Си в качестве инструментального языка позволит создать простые реализации алгоритмов.

# Постановка задачи

Необходимо реализовать программу, которая демонстрирует работу рассматриваемых алгоритмов сортировки. Максимальный размер массива не превышает 100\_000 элементов.

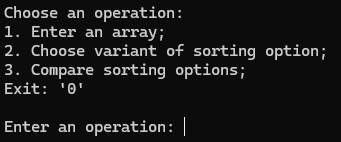
Программа должна поддерживать:

* Выбор способа ввода:
* вручную;
* автозаполнение псевдослучайными числами;
* Проверку корректности ввода данных.
* Выбор метода сортировки: для каждого метода вычисляется количество сравнений, перестановок элементов массива, общее время работы алгоритма в микросекундах.
* Возможность повторить сортировку ранее введенного массива при помощи другого метода сортировки.
* Возможность ввести новый массив.
* Сравнение работы алгоритмов сортировки по времени работы и количеству операций.

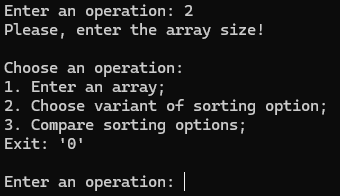
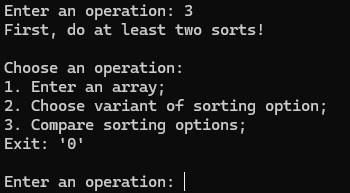
# Руководство пользователя

При запуске программы, высвечивается консоль с меню, где перечислены доступные опции.

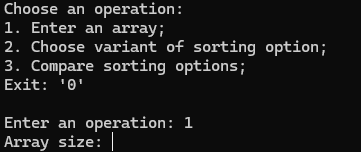
От пользователя ожидается ввод номера действия:



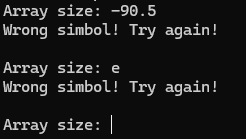
В случае, если пользователь пытается выбрать способ сортировки или сравнение сортировок до того, как был введен массив, то программа выведет предупреждение и предложит ввести номер операции повторно:

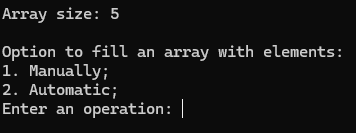
При выборе первого пункта меню – «1. Enter an array;» - программа предложит ввести размер массива:



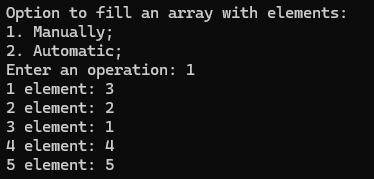
При вводе числа, не являющегося натуральным, или вводе нечисла программа выдаст ошибку и попросит повторить ввод:



После корректного ввода данных программа выведет меню, где предложит два способа заполнения массива: вручную («1. Manually;») и автоматически («2. Automatic;»):

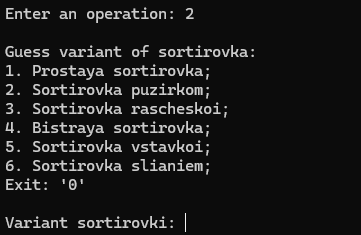


При выборе первого пункта (вручную), пользователь должен ввести каждый элемент массива:



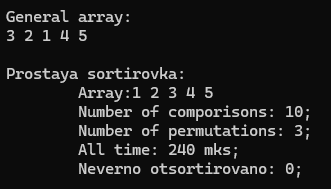
Автоматическое заполнение (пункт 2) осуществляется путем считывания элементов из файла, с записанными псевдослучайными числами.

Если пользователь выберет пункта «2. Choose variant of sorting option;», то будет выведено меню выбора метода сортировки:

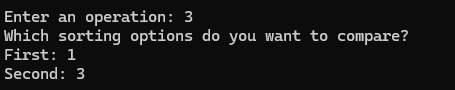


После этого будет создана копия массива, что даёт возможность сравнить несколько сортировок между собой, так как массив-копия перед сортировкой обновляется, чтобы соответствовать исходному.

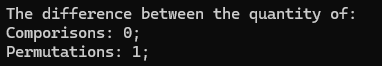
При выборе любого метода сортировки выведется исходный массив, а также следующая информация: отсортированный массив, количество сравнений, количество перестановок элементов и время выполнения в микросекундах:



Как только пользователь отсортировал массив хотя бы двумя разными методами, появится возможность сравнить сортировки между собой. Для этого пользователю необходимо выбрать в начальном меню третий пункт «3. Compare sorting options;». После чего указать первую сортировку и вторую, чтобы сравнить их данные (для примера взяты Простая сортировка и сортировка Шелла [Расчёска]). Для выхода из меню сортировок необходимо ввести “0” (ноль).



Далее будут выведены разница в количестве сравнений и перестановок:



При вводе “0” (нуля) программа завершит выполнение.

# Руководство программиста

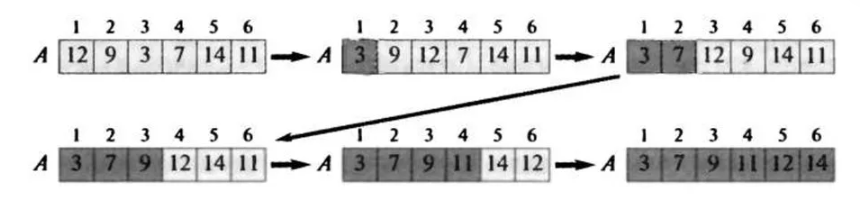
* 1. ***Описание алгоритмов***

Для всех методов сортировки рассматривается сортировка по возрастанию.

1. **Простая сортировка**

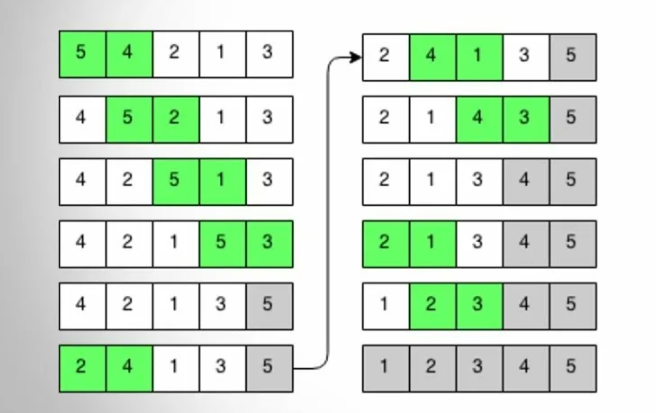
Это алгоритм сортировки, который работает путем многократного прохождения по массиву и сравнения текущего элемента с остальными из неотсортированной части.

При первом проходе выбирается первый элемент массива и сравнивается со всеми остальными. В случае, когда находится элемент, который по значению меньше текущего, элементы поменяются друг с другом значениями, после чего сортировка продолжается. Как только массив закончится, начнется второй проход, но первый элемент будет уже отсортирован. Поэтому сортировка начнется со следующего – второго элемента. Он так же будет сравниваться с оставшимися элементами в случае, если найдется элемент меньше, поменяется с ним значением. Далее сортируется третий элемент и т.д.



1. **Алгоритм сортировки методом пузырька**

Данный алгоритм сравнивает между собой два соседних элемента, и в случае, когда их расположение не соответствует правильному (отсортированному) порядку, меняет их местами. Т.е. сначала сравниваются 1-й и 2-й элементы, потом 2-й и 3-й, затем 3-й и 4-й и т.д. Для полной сортировки приходится совершать несколько проходов по всему массиву и проверять одни и те же элементы.

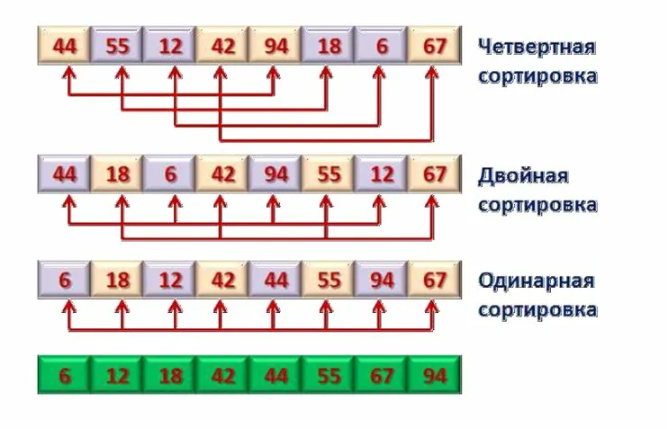


Как можно заметить из примера, самый большой элемент переместился в конец массива при первом же проходе, т.е. его уже сортировать не надо. При следующем проходе в конец массива уйдет второй самый большой элемент и займет предпоследнее место, теперь уже два последних элемента не нуждаются в сортировке. При третьем проходе в конец уйдет третий самый большой элемент и уже три последних элемента не нужно сортировать и т.д.

1. **Сортировка расчёской**

Эта сортировка может рассматриваться как усовершенствованный метод сортировки пузырьком, так как в ней решается проблема перемещения «средних» элементов в массиве. Эффективность сортировки не зависит от величины элементов массива: их модуль может быть мал, огромен или близок к медианному/среднему значению, но перемещаться по массиву они будут с одинаковой скоростью.

В начале выбирается интервал – элементы, отдаленные друг от друга на этот интервал, и будут сравниваться (интервал в примере изначально равен половине длины массива). Выбирается первый элемент и элемент, отдаленный от него на половину длины массива. Если они стоят не в порядке возрастания, то их значения меняются местами. Переходим ко второму элементу массива и делаем тоже самое. Процесс заканчивается, когда добавление интервала вызывает выход за пределы массива.

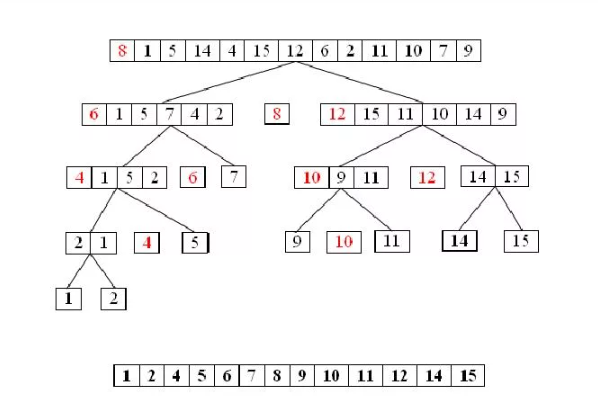


Теперь уменьшаем интервал (в примере - в два раза). Выбираем первый элемент массива и сравниваем с элементом, отдаленным от него на интервал. Меняем местами, если они стоят не в порядке возрастания. Как и при первом проходе делаем данное до тех пор, пока добавление интервала не вызовет выход за приделы массива.

Затем снова уменьшаем интервал, проверяем порядок элементов и т.д. Когда интервал станет равен единице, по сути, будут сравниваться соседние элементы – это последний проход, после которого массив уже полностью отсортирован.

1. **Быстрая сортировка**

Идея алгоритма заключается в том, чтобы во время каждого прохода выбирать из массива опорный элемент и сравнивать с ним все остальные. Данный элемент может выбираться любым образом. Элементы, что меньше опорного ставятся слева от него, те, что больше или равны – справа.

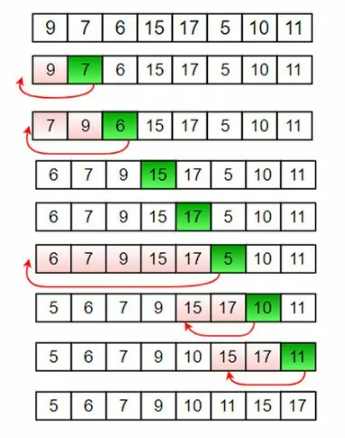


Для каждого из получившихся блоков массива проводится та же операция. Т.е. в каждом блоке выбирается опорный элемент и происходит разбиение. Так до тех пор, пока получившиеся блоки не будут состоять из одного элемента.

Так как снова и снова происходит разбиение частей массива и их сортировка, то алгоритм рекурсивный, то есть вызывает сам себя.

1. **Сортировка вставкой**

Суть алгоритма заключается в том, чтобы разбить массив на две части: отсортированную и неотсортированную – и каждый элемент во время сортировки перемещать из неотсортированной части в отсортированную на «правильную позицию». Поскольку одна часть массива отсортирована, то в ней довольно легко найти «правильную позицию» элемента.



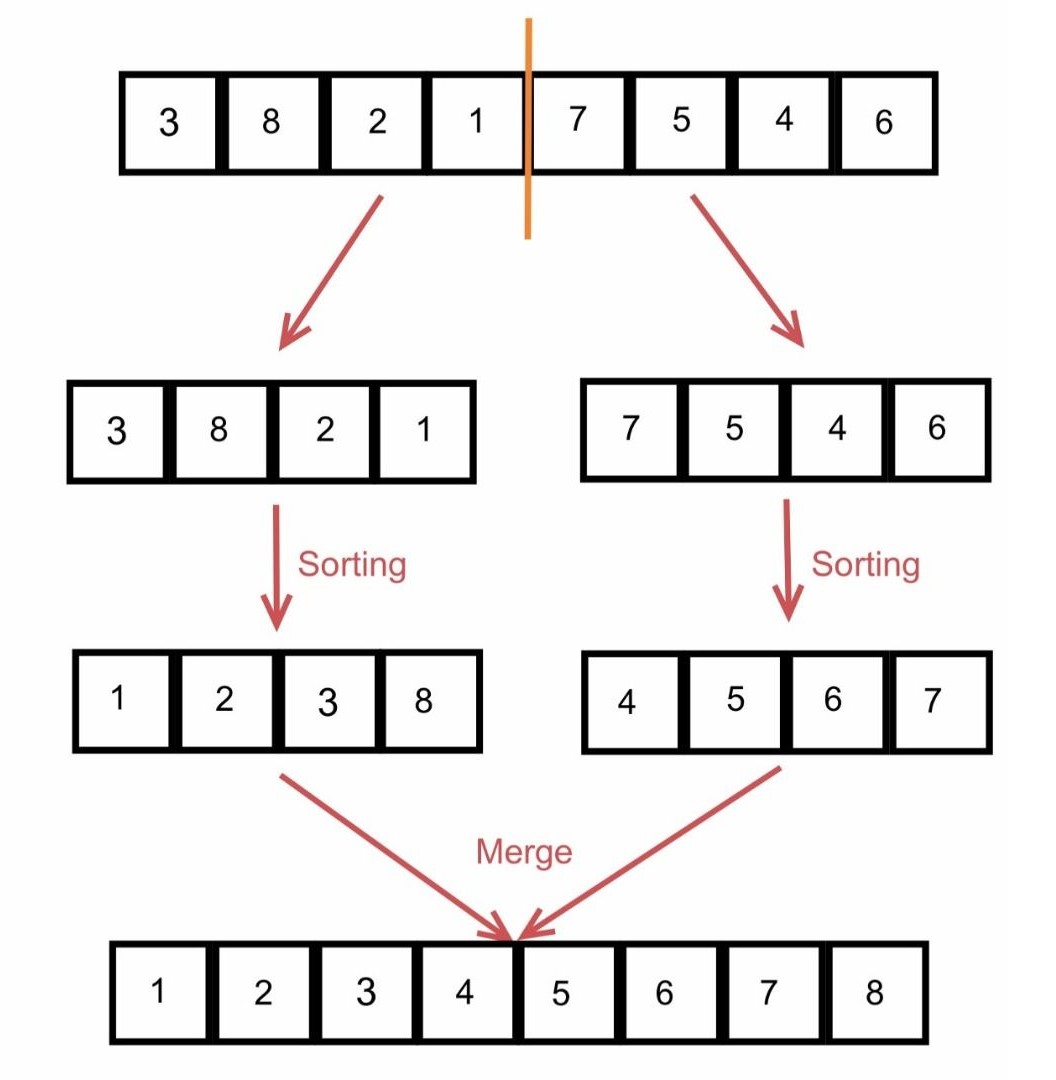
Сортировка начинается со второго элемента, так как первый элемент уже отсортирован. Сравним второй элемент с предыдущим, если он меньше, чем предыдущий, то меняем местами их значения. Теперь отсортированных элементов два. Переходим к следующему элементу и переставляем его в нужное место в отсортированной части.

Определение правильного места в отсортированной части определяется следующим образом. Сначала сравниваем выбранный элемент с предыдущим (начиная с этого «предыдущего» элемента начинается отсортированная часть массива). Если он меньше предыдущего, то сравниваем его с элементом, предшествующем последнему. Снова меньше – сдвигаемся на один шаг влево и снова сравниваем (уже с элементом, предшествующим предшествующему). Так до тех пор, пока не наткнёмся на начало массива. Если выбранный элемент меньше первого элемента массива, то он новый первый элемент.

Процесс продолжается пока неотсортированная часть не окажется пустой.

1. **Сортировка слиянием**

Идея алгоритма заключается в разделении массива пополам, а затем рекурсивном делении каждой из образовавшихся частей до тех пор, пока не получатся части, состоящие из одного элемента. После этого отсортированные подмассивы постепенно объединяются в один упорядоченный массив путем сравнения элементов и подстановки их в правильном порядке.



На этапе объединения используется временный массив. В него отправляются элементы, оказавшиеся наименьшими после попарного сравнения элементов подмассивов.

* 1. ***Описание структур данных***

*void prost(int\* arr, int len)*

Подпрограмма простой сортировки. На вход подаются массив и длина массива. Результатом является отсортированный массив. Также подпрограмма возвращает количество совершенных сравнений и перестановок.

*void puzir(int\* arr, int len)*

Подпрограмма сортировки пузырьком. На вход принимается массив, длина массива. Результатом подпрограммы является отсортированный массив. Также подпрограмма возвращает количество совещенных сравнений и перестановок.

*void vstavka(int\* arr, int len)*

Подпрограмма сортировки вставкой. На вход принимается массив, длина массива. Результатом подпрограммы является отсортированный массив. Также подпрограмма возвращает количество совещенных сравнений и перестановок.

*void slianie(int Lmass[], int Llen, int Rmass[], int Rlen, int Srtd[])*

Подпрограмма сортировки слиянием. На вход принимается левая часть массива, длина левой части массива, правая часть массива, её длина, временный массив. Результатом подпрограммы является отсортированный массив. Также подпрограмма возвращает количество совещенных сравнений и перестановок.

*void SlSort(int arr[], int len)*

Подпрограмма сортировки слиянием. На вход принимается массив, длина массива. Результатом подпрограммы является разбиение массива на две части.

*void rascheska(int\* arr, int len)*

Подпрограмма сортировки расчёсткой. На вход принимается массив, длина массива. Результатом подпрограммы является отсортированный массив. Также подпрограмма возвращает количество совещенных сравнений и перестановок.

*void quickSort(int\* arr, int left, int right)*

Подпрограмма сортировки пузырьком. На вход принимается массив, индекс левого элемента, индекс правого элемента. Результатом подпрограммы является отсортированный массив. Также подпрограмма возвращает количество совещенных сравнений и перестановок.

* 1. ***Описание модульной структуры программы***

*foutput()*

Подпрограмма печати начального меню. Входные и выходные элементы отсутствуют.

*Output()*

Подпрограмма печати меню выбора метода сортировки. Не имеет входных и выходных элементов.

*SortVar(int\* arrCopy, int len, int ch)*

Подпрограмма, запускающая ту сортировку, которую выбрал пользователь. На вход принимается массив, его длина и параметр, показывающий какой метод сортировки выбрал пользователь.

# Вычислительные эксперименты

Для того чтобы сравнить методы сортировки по количеству операций и времени работы, необходимо отсортировать один и тот же массив разными методами. Для вычислительных экспериментов было решено выбрать массивы размеров 1000, 5000, 10000, 15000.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Время сортировки | | | | | | |
| Размер массива | Простая сортировка | Сортировка пузырьком | Сортировка расчёской | Быстрая сортировка | Сортировка вставкой | Сортировка слиянием |
| 1000 | 4102 | 3163 | 597 | 602 | 1783 | 833 |
| 5000 | 35047 | 35994 | 1338 | 1557 | 30167 | 4767 |
| 10000 | 125982 | 134439 | 3286 | 2513 | 93069 | 7207 |
| 15000 | 215988 | 303478 | 3783 | 3151 | 238819 | 10569 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Количество операций сравнения | | | | | | |
| Размер массива | Простая сортировка | Сортировка пузырьком | Сортировка расчёской | Быстрая сортировка | Сортировка вставкой | Сортировка слиянием |
| 1000 | 499500 | 499500 | 23018 | 5250 | 196017 | 9101 |
| 5000 | 12497500 | 12497500 | 152491 | 30126 | 5959995 | 57931 |
| 10000 | 49995000 | 49995000 | 339575 | 64937 | 24327973 | 125896 |
| 15000 | 112492500 | 112492500 | 535560 | 109795 | 74273942 | 173922 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Количество перемещений | | | | | | |
| Размер массива | Простая сортировка | Сортировка пузырьком | Сортировка расчёской | Быстрая сортировка | Сортировка вставкой | Сортировка слиянием |
| 1000 | 102545 | 195024 | 60053 | 3490 | 195024 | 8594 |
| 5000 | 797290 | 5955002 | 93428 | 22164 | 5955002 | 55170 |
| 10000 | 1678181 | 24317980 | 107820 | 49489 | 24317980 | 120306 |
| 15000 | 3384774 | 74258950 | 157888 | 86907 | 74258950 | 163748 |

# Заключение

В результате выполнения данной работы была написана программа, анализирующая методы сортировки: простая сортировка, пузырьком, расчёской, быстрая сортировка, вставкой, слиянием. Программа обладает высоким параметром быстродействия, относительно маленьким размером и не требовательна к системным ресурсам компьютера. В качестве недостатка программы можно отнести то, что точность выполнения программы зависит от тактовой частоты компьютера. Этот недостаток можно решить путём многократного замерения времени выполнения и вычисления среднего.

При выполнении работы удалось рассмотреть и проанализировать лишь небольшую часть основных алгоритмов сортировки данных: рассмотрены некоторые простые алгоритмы (сюда относятся «пузырек», вставки) и усовершенствованные (сортировка расчёской, быстрая сортировка и сортировка слиянием).

Общим недостатком простых алгоритмов является их слишком медленная работа на больших массивах данных. Это подтверждается вычислительными исследованиями. Усовершенствованные алгоритмы работают гораздо быстрее, причем на очень больших массивах различие может достигать многих тысяч раз.