**Муниципальное бюджетное учреждение – гимназия №19 имени Героя Советского Союза В.И. Меркулова г. Орла**

**ИНДИВИДУАЛЬНЫЙ ПРОЕКТ**

**Тема: "** **Исследование способов сортировки, их сравнение и визуализация."**

**Вид проекта: прикладной**

**Автор проекта:**

**обучающийся 10 Е класса**

**Мальцев Кирилл Леонидович**

**Руководитель проекта:**

**Учитель (информатики)**

**Кривоногов Константин Юрьевич**

**г. Орёл**

**2022-2023**

Оглавление.

[**Введение**. 4](#_Toc131754238)

[**Основная часть.** 6](#_Toc131754239)

[1. Определение 6](#_Toc131754240)

[2. История и концепции. 6](#_Toc131754241)

[3. Виды алгоритмов сортировки. 8](#_Toc131754242)

[3.1. Exchange Sorts (сортировки обменом) 8](#_Toc131754243)

[3.2. Selection Sorts (сортировки выбором) 8](#_Toc131754244)

[3.3. Insertion Sorts (сортировки вставками) 8](#_Toc131754245)

[3.4. Merge Sorts (сортировки слиянием) 8](#_Toc131754246)

[3.5. Distribution Sorts (сортировки распределением) 9](#_Toc131754247)

[3.6. Hybrid Sorts (гибридные сортировки) 9](#_Toc131754248)

[4. Характеристики алгоритмов сортировки. 10](#_Toc131754249)

[4.1. Общий метод. 10](#_Toc131754250)

[4.2. Время. 10](#_Toc131754251)

[4.3. Вычислительная сложность. 10](#_Toc131754252)

[4.4. Память. 10](#_Toc131754253)

[4.5. Рекурсия. 10](#_Toc131754254)

[4.6. Стабильность. 11](#_Toc131754255)

[4.7. Адаптивность. 11](#_Toc131754256)

[4.8. Сфера применения. 11](#_Toc131754257)

[5. Стабильность. 11](#_Toc131754258)

[6. Рассмотренные алгоритмы сортировки. 12](#_Toc131754259)

[6.1. Пузырьковая сортировка. 12](#_Toc131754260)

[6.2. Сортировка перемешиванием. 13](#_Toc131754261)

[6.3. Сортировка вставками. 14](#_Toc131754262)

[6.4. Гномья сортировка. 14](#_Toc131754263)

[6.5. Сортировка выбором. 15](#_Toc131754264)

[6.6. Сортировка чёт-нечёт. 15](#_Toc131754265)

[6.7. Сортировка расчёской. 16](#_Toc131754266)

[6.8. Сортировка Шелла. 17](#_Toc131754267)

[6.9. Быстрая сортировка. 18](#_Toc131754268)

[6.10. Сортировка слиянием. 19](#_Toc131754269)

[6.11. Пирамидальная сортировка. 21](#_Toc131754270)

[6.12. Timsort. 22](#_Toc131754271)

[6.13. Introsort. 23](#_Toc131754272)

[6.14. Сортировка подсчётом. 23](#_Toc131754273)

[6.15. Поразрядная сортировка. 24](#_Toc131754274)

[7. Способ тестирования алгоритмов. 25](#_Toc131754275)

[8. Сравнение поведения алгоритмов при изменении размера массива. 26](#_Toc131754276)

[9. Асимптотика и затраты памяти, общая классификация. 27](#_Toc131754277)

[**Заключение.** 28](#_Toc131754278)

[**Список источников.** 29](#_Toc131754279)

# **Введение**.

Алгоритмы и структуры данных являются основой компьютерного программирования. Знание этих структур и алгоритмов позволяет осуществлять выбор наиболее оптимальных способов решения задач, возникающих при создании программного обеспечения различного назначения.

Одним из важнейших процедур обработки структурированной информации является сортировка. Сортировкой называют процесс перегруппировки заданной последовательности (кортежа) объектов в некотором определенном порядке. Определенный порядок (например, упорядочение в алфавитном порядке, по возрастанию или убыванию количественных характеристик, по классам, типам и т.п.) в последовательности объектов необходим для удобства работы с этими объектами.

Актуальность: в алгоритмизации и программировании сортировки необходимы для решения многих задач или подзадач. К примеру, они упрощают реализацию поиска и извлечения элементов из списка, что необходимо в реализации решений различных проблем. Данный проект будет полезен тем, что поможет разобраться в работе данных алгоритмов, путём их визуализации. Также алгоритмы сортировки преобладают на начальных занятиях по информатике, где обилие алгоритмов для решения задачи обеспечивает введение в различные основные концепции алгоритмов, такие как нотация большого O, алгоритмы «разделяй и властвуй», структуры данных (кучи и деревья), анализ наилучшего, наихудшего и среднего случаев, а также компромиссы между временем и пространством.

Цель: с помощью языка программирования Python и событийно-ориентированной графической библиотеки Tkinter создать приложение для визуализации процесса сортировки и оценки затраченного времени и памяти работы алгоритмов.

Задачи:

1. Исследовать различные группы алгоритмов сортировки и систематизировать полученную информацию.
   1. Exchange Sorts (сортировки обменом)
   2. Selection Sorts (сортировки выбором)
   3. Insertion Sorts (сортировки вставками)
   4. Merge Sorts (сортировки слиянием)
   5. Distribution Sorts (сортировки распределением)
   6. Hybrid Sorts (гибридные сортировки)
2. Реализовать визуализацию алгоритмов сортировки (используя графическую библиотеку Tkinter создать программу с пользовательским интерфейсом, с помощью которой будет показан принцип работы данных алгоритмов).
3. Найти и проанализировать асимптотику данных алгоритмов.
4. Подвести итоги и выбрать наиболее оптимальные виды сортировок, на основе полученных данных.

# **Основная часть.**

1. Определение

Алгоритм сортировки – это алгоритм упорядочивания элементов списка. Упорядочивание производится в соответствии со значением ключа сортировки по возрастанию или убыванию его значения.

1. История и концепции.

Первые прототипы современных методов сортировки появились уже в XIX веке. К 1890 году для ускорения обработки данных переписи населения в США американец Герман Холлерит создал первый статистический табулятор — электромеханическую машину, предназначенную для автоматической обработки информации, записанной на перфокартах. Работа сортировальной машины Холлерита основывалась на методах поразрядной сортировки.

В дальнейшем история алгоритмов оказалась связана с развитием электронно-вычислительных машин. По некоторым источникам, именно программа сортировки стала первой программой для вычислительных машин. В 1945 году Джон фон Нейман для тестирования ряда команд для EDVAC разработал программы сортировки методом слияния. В том же году немецкий инженер Конрад Цузе разработал программу для сортировки методом простой вставки. К этому времени уже появились быстрые специализированные сортировальные машины, в сопоставлении с которыми и оценивалась эффективность разрабатываемых ЭВМ.

Позже было предложено множество различных алгоритмов сортировки: например, слияние с вставкой, обменная поразрядная сортировка, каскадное слияние и метод Шелла в 1959 году, многофазное слияние в 1960 году, быстрая сортировка Хоара в 1962 году, пирамидальная сортировка Уильямса и обменная сортировка со слиянием Бэтчера в 1964 году. В конце 60-х годов произошло и интенсивное развитие теории сортировки. Появившиеся позже алгоритмы во многом являлись вариациями уже известных методов. Получили распространение адаптивные методы сортировки, ориентированные на более быстрое выполнение в случаях, когда входная последовательность удовлетворяет заранее установленным критериям. Новые алгоритмы изобретаются до сих пор, широко используемый Timsort датируется 2002 годом, а библиотека sort впервые опубликован в 2006 году.

1. Задача алгоритмов сортировки.
   1. Задача сортировки состоит в том, что в произвольной последовательности элементов списка, расположить их таким образом, чтобы каждое последующее значение ключа сортировки было больше предыдущего, в случае сортировки по возрастанию. Если производится сортировка по убыванию, каждое последующее значение ключа, соответственно, должно быть меньше предыдущего.
   2. Более формально: требуется упорядочить N элементов:R1, R2, …, Rn, где каждый элемент представляет из себя запись Ri, содержащую некоторую информацию и ключ Ki, управляющий процессом сортировки. На множестве ключей определено отношение порядка «<» так, чтобы для любых трёх значений ключей a, b, c выполнялись следующие условия:

* закон трихотомии: либо a<b, либо a>b, либо a=b;
* закон транзитивности: если a<b и b<c, то a<c.
  1. Данные условия определяют математическое понятие линейного или совершенного упорядочения, а удовлетворяющие им множества поддаются сортировке большинством методов. Т.е. задачей сортировки является нахождение такой перестановки записей p(1), p(2), …, p(n) с индексами {1,2,…,N}, после которой ключи расположились бы в порядке неубывания: Kp(1) ≤ Kp(2) ≤ … ≤ Kp(n)
  2. Алгоритмы сортировки сравнением имеют фундаментальное требование n log n сравнений или кратного n log n сравнений, где n — количество элементов в массиве для сортировки.

## Виды алгоритмов сортировки.

### Exchange Sorts (сортировки обменом)

Алгоритм сортировки прямым обменом основан на принципе сравнения и обмена пары соседних элементов до тех пор, пока не будут отсортированы все элементы. Данный метод можно разделить на 3 этапа:

1. Попарно сравниваются элементы массива.

2. Если элемент слева больше элемента справа, то элементы меняются местами.

3. Повторяем пункты 1-2 до тех пор, пока массив не отсортируется.

* 1. Selection Sorts (сортировки выбором)

Данный метод также можно разделить на 3 этапа:

1. В неотсортированном подмассиве ищется локальный максимум (минимум).
2. Найденный максимум (минимум) меняется местами с последним (первым) элементом в подмассиве.
3. Повторяем пункты 1-2 до тех пор, пока массив не отсортируется.
   1. Insertion Sorts (сортировки вставками)

Суть его заключается в том, что на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для него нужную позицию и перемещаем на неё элемент.

* 1. Merge Sorts (сортировки слиянием)

Алгоритм использует принцип «разделяй и властвуй»: задача разбивается на подзадачи меньшего размера, которые решаются по отдельности, после чего их решения комбинируются для получения решения исходной задачи. Конкретно процедуру сортировки слиянием можно описать следующим образом:

1. Если в рассматриваемом массиве один элемент, то он уже отсортирован — алгоритм завершает работу.
2. Иначе массив разбивается на две части, которые сортируются рекурсивно.
3. После сортировки двух частей массива к ним применяется процедура слияния, которая по двум отсортированным частям получает исходный отсортированный массив.
   1. Distribution Sorts (сортировки распределением)

В сортировках распределением элементы распределяются и перераспределяются по классам до тех пор, пока массив не отсортируется.

В самом общем случае это происходит по примерно одинаковой схеме. Элементы разбрасываются по классам по какому-либо признаку. Если это не привело к упорядочиванию массива, то происходит уточнение признаков принадлежности к классу и элементы раскидываются по уточнённым классам снова. И так происходит до тех пор, пока массив не станет упорядоченным.

В сортировках распределением почти всегда отсутствуют сравнения элементов между собой и их обмены. Главное — это принадлежит ли элемент к некоторому классу или нет, его сравнение с другими элементами редко играет роль.

Обычно у этих сортировок линейная сложность по времени (а не логарифмическая, как у эффективных сортировок обменами, слиянием, выбором или вставками). Также алгоритмы этого класса почти всегда требуют много дополнительной памяти, поскольку сгруппированные по классам элементы приходится где-то хранить.

* 1. Hybrid Sorts (гибридные сортировки)

В основу сортировки могут быть положены обмены, вставки, выбор, слияние и распределение. Но если в алгоритме комбинируются разные методы, то тогда он относится к классу гибридных сортировок.

## Характеристики алгоритмов сортировки.

* 1. Общий метод.

Вставка, обмен, выбор, слияние и т. д.

* 1. Время.

Время – главный параметр, показывающий скорость упорядочивания элементов от исходного произвольного порядка к конечному результату. В основном, на время, влияет количество шагов, за которое алгоритм придет к результату. Под шагами подразумевается количество сравнений и замен элементов. Причем скорость упорядочивания одного вида алгоритма может существенно отличатся для разных наборов сортируемых данных. Различают худший, средний и лучший результат работы алгоритма.

* 1. Вычислительная сложность.

Лучшее, худшее и среднее поведение в зависимости от размера списка. Для типичных алгоритмов последовательной сортировки хорошее поведение — O( n log n ), а плохое поведение — O( n 2 ). Идеальным поведением для последовательной сортировки является O( n ), но в среднем это невозможно.

* 1. Память.

Память – занимаемое дополнительное место в памяти. Некоторые алгоритмы для своей работы дублируют исходный массив или его части. Учитывая, что сам массив может быть очень большим, то затраты памяти на работу таких алгоритмов могут быть очень существенными, а иногда могут являться и основным критерием при выборе алгоритма сортировки. Память, затрачиваемая на хранение исходного массива, в учет этого критерия не берется.

* 1. Рекурсия.

Некоторые алгоритмы либо рекурсивны, либо нерекурсивны.

* 1. Стабильность.

Стабильные алгоритмы сортировки поддерживают относительный порядок записей с одинаковыми ключами (т. е. значениями)

* 1. Адаптивность.

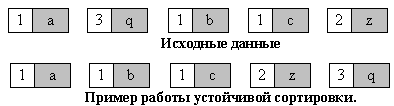
Влияет ли предварительная сортировка ввода на время выполнения. Известно, что алгоритмы, учитывающие это, являются адаптивными.

* 1. Сфера применения.

Определяет назначение алгоритмов и характеризуется двумя типами сортировки – внутренней и внешней. Внутренняя сортировка, т.е. работа с массивом данных, целиком расположенном в памяти и возможным доступом к его любому элементу. Внешняя сортировка характеризуется работой с данными большого объема и последовательным доступом к отдельным элементам списка. Применяется, например, при упорядочении файлов.

## Стабильность.

Как уже было сказано выше: стабильные алгоритмы сортировки поддерживают относительный порядок записей с одинаковыми ключами (т. е. значениями)



Алгоритмы стабильной сортировки сортируют одинаковые элементы в том же порядке, в котором они появляются во входных данных. Например, в данном примере сортировки относительное расположение буквенных ключей не изменилось. Это позволяет использовать несколько разных правильно отсортированных версий исходного списка. Алгоритмы стабильной сортировки выбирают один из них в соответствии со следующим правилом: если два элемента сравниваются как равные, то их относительный порядок будет сохранен, т.е. перед другим на выходе.

Более формально сортируемые данные могут быть представлены в виде записи или кортежа значений, а часть данных, которая используется для сортировки, называется ключом. В данном примере элементы представлены записью (число, буква), а ключ — число. Алгоритм сортировки стабилен, если всякий раз, когда есть две записи R и S с одним и тем же ключом, и R появляется перед S в исходном списке, R всегда будет стоять перед S в отсортированном списке.

Когда равные элементы неразличимы, например, с целыми числами или, в более общем случае, с любыми данными, где весь элемент является ключом, стабильность не является проблемой. Стабильность также не является проблемой, если все ключи разные.

1. Рассмотренные алгоритмы сортировки.
   1. **Пузырьковая сортировка.**

Сортировка простыми обменами, сортировка пузырьком (англ. bubble sort) — простой алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов.

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов. Проходы по массиву повторяются раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

Используемый метод: обмен

Сфера применения: Алгоритм считается учебным и практически не применяется вне учебной литературы, вместо него на практике применяются более эффективные алгоритмы сортировки. В то же время метод сортировки обменами лежит в основе некоторых более совершенных алгоритмов, таких как шейкерная сортировка, пирамидальная сортировка и быстрая сортировка.

* 1. Сортировка перемешиванием.

Сортировка перемешиванием, или Шейкерная сортировка, или двунаправленная (англ. Shaker sort) — разновидность пузырьковой сортировки.

Алгоритм ялвяется модификацией пузырьковой сортировки и реализуется, если заметить 2 факта:

* Во-первых, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, её можно исключить из рассмотрения.
* Во-вторых, при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо.

Эти две идеи приводят к следующим модификациям в методе пузырьковой сортировки. Границы рабочей части массива (то есть части массива, где происходит движение) устанавливаются в месте последнего обмена на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

Используемый метод: обмен

Сфера применения: Аналогично пузырковой сортировке алгоритм считается учебным и практически не применяется вне учебной литературы, вместо него на практике применяются более эффективные алгоритмы сортировки.

* 1. Сортировка вставками.

Сортировка вставками (англ. Insertion sort) — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

Используемый метод: вставка

Сфера применения: сортировка вставками стабильна, поскольку сохраняет порядок одинаковых элементов во входном массиве. Это важно для многих приложений, таких как поиск данных или финансовый анализ.

* 1. Гномья сортировка.

Гномья сортировка (англ. Gnome sort) — алгоритм сортировки, похожий на сортировку вставками, но в отличие от последней перед вставкой на нужное место происходит серия обменов, как в сортировке пузырьком. Название происходит от предполагаемого поведения садовых гномов при сортировке линии садовых горшков.

Алгоритм гномьей сортировки разработан, по словам официального автора(Дика Груна), гномами, которые сортировали садовые горшки. Правда это или нет, но алгоритм очень прост, особенно для начинающих. По сути, в алгоритме сравниваются рядом стоящие горшки, если они стоят в нужном порядке, тогда мы переходим на следующий элемент массива, если нет, ты мы их переставляем и переходим на предыдущий. Нету предыдущего элемента — идём вперед, нету следующего — значит мы закончили. Изначально мы находимся на втором элементе массива.

Используемый метод: обмен

Сфера применения: из-за сложности n2 алгоритм не имеет смысла применять в реальных задачах. Является ознакомительным.

* 1. Сортировка выбором.

Сортировка выбором (Selection sort) — алгоритм сортировки. Может быть как устойчивый, так и неустойчивый.

Шаги алгоритма:

1. находим номер минимального значения в текущем списке
2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

Используемый метод: выбор.

Сфера применения: из-за сложности n2 алгоритм не имеет смысла применять в реальных задачах, но его принцип лежит в основе многих других более быстрых алгоритмов. Является ознакомительным.

* 1. Сортировка чёт-нечёт.

Этот относительно простой алгоритм сортировки, разработанный для использования на параллельных процессорах, является модификацией пузырьковой сортировки. Суть модификации в том, чтобы сравнивать элементы массива под чётными и нечётными индексами с последующими элементами независимо.

Заводится флаг, определяющий отсортирован ли массив. В начале итерации ставится в состояние «истина», далее каждый нечётный элемент сверяется с последующим и если они стоят в неправильном порядке (предыдущий больше следующего), то они меняются местами, и флаг ставится в состояние «ложь». То же самое делается с чётными элементами. Алгоритм не прекращает работу, пока флаг не останется в состоянии «истина».

Используемый метод: обмен.

Сфера применения: этот относительно простой алгоритм сортировки, разработанный для использования на параллельных процессорах.

* 1. Сортировка расчёской.

Сортировка расчёской (англ. comb sort) — это довольно упрощённый алгоритм сортировки. Сортировка расчёской улучшает сортировку пузырьком, и конкурирует с алгоритмами, подобными быстрой сортировке. Основная идея — устранить черепах, или маленькие значения в конце списка, которые крайне замедляют сортировку пузырьком (кролики, большие значения в начале списка, не представляют проблемы для сортировки пузырьком).

В сортировке пузырьком, когда сравниваются два элемента, промежуток (расстояние друг от друга) равен 1. Основная идея сортировки расчёской в том, что этот промежуток может быть гораздо больше, чем единица.

В «пузырьке», «шейкере» и «чёт-нечете» при переборе массива сравниваются соседние элементы. Основная идея «расчёски» в том, чтобы первоначально брать достаточно большое расстояние между сравниваемыми элементами и по мере упорядочивания массива сужать это расстояние вплоть до минимального. Таким образом, мы как бы причёсываем массив, постепенно разглаживая на всё более аккуратные пряди. Первоначальный разрыв между сравниваемыми элементами лучше брать с учётом специальной величины, называемой фактором уменьшения, оптимальное значение которой равно примерно 1,247. Сначала расстояние между элементами максимально, то есть равно размеру массива минус один. Затем, пройдя массив с этим шагом, необходимо поделить шаг на фактор уменьшения и пройти по списку вновь. Так продолжается до тех пор, пока разность индексов не достигнет единицы. В этом случае сравниваются соседние элементы как и в сортировке пузырьком, но такая итерация одна.

Используемый метод: обмен.

Сфера применения: может конкурировать по скорости со многими быстрыми алгоритмами (быстрая сортировка, сортировка слиянием и т.д.)

* 1. Сортировка Шелла.

Сортировка Шелла (англ. Shell sort) — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Также как и в сортировке расчёской идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии. После этого процедура повторяется для некоторых меньших значений, а завершается сортировка Шелла упорядочиванием элементов обычной сортировкой вставками. Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

* отсутствие потребности в памяти под стек;
* отсутствие деградации при неудачных наборах данных — быстрая сортировка легко деградирует до O(n²), что хуже, чем худшее гарантированное время для сортировки Шелла.

Используемый метод: вставка.

Сфера применения: в некоторых случаях может конкурировать с более быстрыми сортировками.

* 1. Быстрая сортировка.

Быстрая сортировка, сортировка Хоара (англ. quicksort), часто называемая qsort (по имени в стандартной библиотеке языка Си) — алгоритм сортировки, разработанный английским информатиком Тони Хоаром во время своей работы в МГУ в 1960 году.

Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем n log n обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «Пузырьковая сортировка» и «Шейкерная сортировка»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы (таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов).

Общая идея алгоритма состоит в следующем:

* Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность (см. ниже).
* Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
* Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

Используемый метод: разделение, обмен.

Сфера применения: является одним из лучших алгоритмов сортировок, используется для решения многих задач.

* 1. Сортировка слиянием.

Сортировка слиянием (англ. merge sort) — алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один. Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по возрастанию подмассива. Тогда:

3.2. Слияние двух подмассивов в третий результирующий массив. На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на «Прицепление» остатка. Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

Используемый метод: объединение, вставка.

Сфера применения: эта сортировка — хороший пример использования принципа «разделяй и властвуй». Из-за сложности выполнения *n* log *n* может конкурировать с быстрой сортировкой.

* 1. Пирамидальная сортировка.

Пирамидальная сортировка (или сортировка кучей, HeapSort) — это метод сортировки сравнением, основанный на такой структуре данных как двоичная куча. Она похожа на сортировку выбором, где мы сначала ищем максимальный элемент и помещаем его в конец. Далее мы повторяем ту же операцию для оставшихся элементов.

Алгоритм пирамидальной сортировки в порядке по возрастанию:

1. Постройте max-heap из входных данных.
2. На данном этапе самый большой элемент хранится в корне кучи. Замените его на последний элемент кучи, а затем уменьшите ее размер на 1. Наконец, преобразуйте полученное дерево в max-heap с новым корнем.
3. Повторяйте вышеуказанные шаги, пока размер кучи больше 1.

*Как построить кучу?*

Процедура преобразования в кучу (далее процедура heapify) может быть применена к узлу, только если его дочерние узлы уже преобразованы. Таким образом, преобразование должно выполняться снизу вверх. Давайте разберемся с помощью примера:

Входные данные: 4, 10, 3, 5, 1

4(0)

/ \

10(1) 3(2)

/ \

5(3) 1(4)

Числа в скобках представляют индексы в представлении данных в виде массива.

Применение процедуры heapify к индексу 1:

4(0)

/ \

10(1) 3(2)

/ \

5(3) 1(4)

Применение процедуры heapify к индексу 0:

10(0)

/ \

5(1) 3(2)

/ \

4(3) 1(4)

Процедура heapify вызывает себя рекурсивно для создания кучи сверху вниз.

* 1. Timsort.

Timsort — гибридный алгоритм сортировки, сочетающий сортировку вставками и сортировку слиянием, опубликованный в 2002 году Тимом Петерсом. В настоящее время Timsort является стандартным алгоритмом[1] сортировки в Python, OpenJDK 7[2], Apple Swift Standard Library 5[3] и реализован в Android JDK 1.5[4]. Основная идея алгоритма в том, что в реальном мире сортируемые массивы данных часто содержат в себе упорядоченные подмассивы. На таких данных Timsort существенно быстрее многих алгоритмов сортировки[5].

* По специальному алгоритму входной массив разделяется на подмассивы.
* Каждый подмассив сортируется сортировкой вставками.
* Отсортированные подмассивы собираются в единый массив с помощью модифицированной сортировки слиянием.
* Используемый метод: вставка, слияние.
* Сфера применения: показывает отличное время выполнения на частично отсортированных массивах. Лежит в основе библиотеки *«sort»* в языке программирования Python.
  1. Introsort.

Introsort или интроспективная сортировка — алгоритм сортировки, предложенный Дэвидом Мюссером (англ.)рус. в 1997 году. Он использует быструю сортировку и переключается на пирамидальную сортировку, когда глубина рекурсии превысит некоторый заранее установленный уровень (например, логарифм от числа сортируемых элементов). Этот подход сочетает в себе достоинства обоих методов с худшим случаем O(n log n) и быстродействием, сравнимым с быстрой сортировкой. Так как оба алгоритма используют сравнения, этот алгоритм также принадлежит классу сортировок на основе сравнений.

* Используемый метод: разделение и выбор.
* Сфера применения: лежит в основе библиотеки *«sort»* в языке программирования C++.
  1. Сортировка подсчётом.

Сортировка подсчетом — это метод сортировки, основанный на ключах в определенном диапазоне. Он работает путем подсчета количества объектов, имеющих различные значения ключа (своего рода хеширование). Затем выполните некоторые арифметические операции, чтобы вычислить положение каждого объекта в выходной последовательности.

Характеристики счетной сортировки:

* Сортировка с подсчетом делает предположения о данных, например, предполагается, что значения будут находиться в диапазоне от 0 до 10 или от 10 до 99 и т. д. Некоторые другие сортировки с подсчетом предполагают, что входные данные будут действительными числами.
* Как и другие алгоритмы, этот алгоритм сортировки не является алгоритмом, основанным на сравнении, он хеширует значение во временном массиве счетчиков и использует их для сортировки.
* Он использует временный массив, что делает его алгоритмом не на месте.

Используемый метод: разделение.

Сфера применения: используется для сортировки числовых массивов с малым процентом уникальности элементов.

* 1. Поразрядная сортировка.

Поразрядная сортировка (англ. radix sort) — алгоритм сортировки, который выполняется за линейное время. Существуют стабильные варианты.

Исходно предназначен для сортировки целых чисел, записанных цифрами. Но так как в памяти компьютеров любая информация записывается целыми числами, алгоритм пригоден для сортировки любых объектов, запись которых можно поделить на «разряды», содержащие сравнимые значения, например, строки, и вообще любые данные, представленные в виде набора байтов.

Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

Так как выравнивать сравниваемые записи относительно друг друга можно в разную сторону, на практике существуют два варианта этой сортировки. Для чисел они называются в терминах значимости разрядов числа, и получается так: можно выровнять записи чисел в сторону менее значащих цифр (по правой стороне, в сторону единиц — LSD от англ. least significant digit) или более значащих цифр (по левой стороне, со стороны более значащих разрядов — MSD от most significant digit).

При LSD-сортировке получается порядок, уместный для чисел. Например: 1, 2, 9, 10, 21, 100, 200, 201, 202, 210. То есть, здесь значения сначала сортируются по единицам, затем сортируются по десяткам, сохраняя отсортированность по единицам внутри десятков, затем по сотням, сохраняя отсортированность по десяткам и единицам внутри сотен, и так далее.

При MSD-сортировке получается алфавитный порядок, который уместен для сортировки строк текста. Например «b, c, d, e, f, g, h, i, j, ba» отсортируется как «b, ba, c, d, e, f, g, h, i, j». Если MSD применить к числам, то получится алфавитный, но не числовой порядок: 1, 10, 100, 2, 200, 201, 202, 21, 210, 9.

Используемый метод: разделение.

Сфера применения: используется для сортировки числовых массивов с элементами, состоящими из большого количества разрядов.

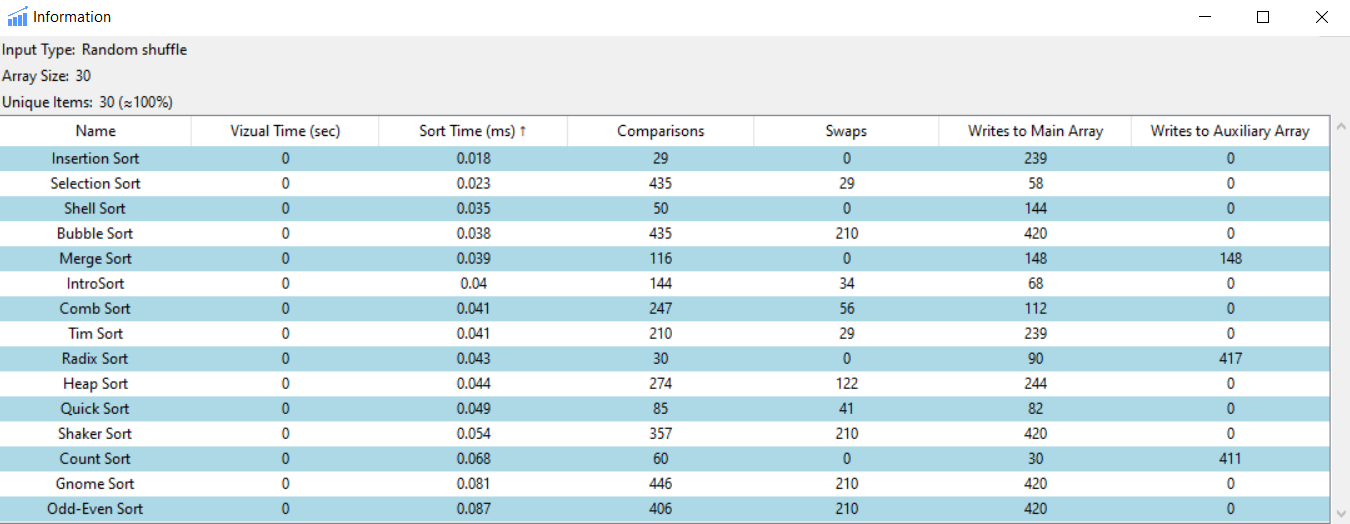
1. Способ тестирования алгоритмов.

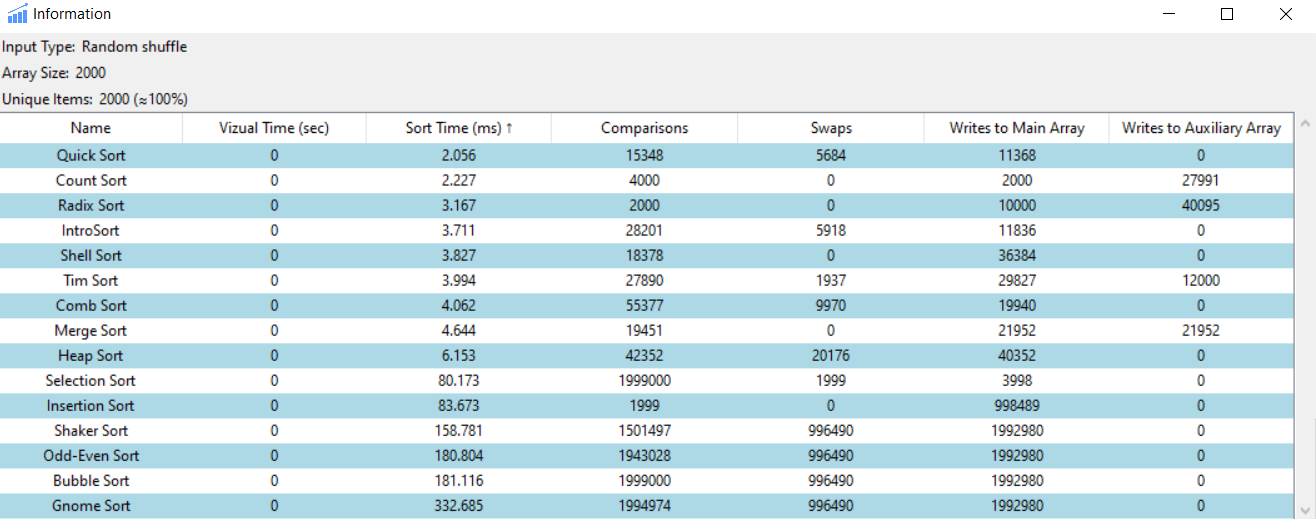
Для измерения времени работы алгоритмов используется модуль timeit в python.



Функции «execute» передаётся исполняемый алгоритм в качестве функции с уже заданными параметрами. Далее он тестируется x раз, где x=10, по умолчанию. В зависимости от выбранного режима сборщик мусора в Python (garbage collector) может быть выключен или выключен, что может незначительно повлиять на итоговый результат исполнения программы.

1. Сравнение поведения алгоритмов при изменении размера массива.





1. Асимптотика и затраты памяти, общая классификация.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Название | Время выполнения | | | Память | Стабильность | Метод |
| Лучшее | Среднее | Худшее |
| Пузырьковая сортировка | *n* | *n2* | *n2* | *1* | Да | Обмен |
| Сортировка перемешиванием | *n* | *n2* | *n2* | *1* | Да | Обмен |
| Сортировка вставками | *n* | *n2* | *n2* | *1* | Да | Вставка |
| Гномья Сортировка | *n* | *n2* | *n2* | *1* | Да | Обмен |
| Сортировка выбором | *n2* | *n2* | *n2* | *1* | Нет | Выбор |
| Сортировка чёт-нечёт | *n* | *n2* | *n2* | *1* | Да | Обмен |
| Сортировка рассчёской | *n log n* | *n2* | *n2* | *1* | Нет | Обмен |
| Сортировка Шелла | *n log n* | *n4/3* | *n3/2* | *1* | Нет | Вставка |
| Быстрая сортировка | *n log n* | *n log n* | *n2* | *log n* | Нет | Разделение |
| Сортировка слиянием | *n log n* | *n log n* | *n log n* | *n* | Да | Объединение |
| Пирамидальная сортировка | *n log n* | *n log n* | *n log n* | *1* | Нет | Разделение и выбор |
| Timsort | *n* | *n log n* | *n log n* | *n* | Да | Вставка и слияние |
| Intosort | *n log n* | *n log n* | *n log n* | *log n* | Нет | Разделение и выбор |
| Сортировка подсчётом | – | *n+r* | *n+r* | *n+r* | Да | Разделение |
| Поразрядная сортировка | *n* | *n\*d* | *n\*d* | *n+d* | Да | Разделение |

# **Заключение.**

В результате исследования были рассмотрены различные методы сортировки, их принципы работы и особенности. Были проведены сравнительные анализы эффективности сортировок, а также произведена визуализация процесса сортировки для лучшего понимания и наглядности.

Выводы показали, что каждый метод сортировки имеет свои преимущества и недостатки, и выбор определенного метода зависит от конкретной задачи и ее условий. Некоторые методы более эффективны на небольших массивах данных, а другие – на больших. Также важно учитывать особенности данных, которые нужно отсортировать. Одним из самых лучших алгоритмов сортировки является быстрая сортировка Хоара. С ней могут конкурировать Timsort и Introsort, а также Merge Sort и, в некоторой степени, сортировка расчёской.

Использование визуализации позволяет наглядно увидеть процесс сортировки и помочь понять, как работает каждый метод. Это может быть полезно не только для студентов и начинающих программистов, но и для опытных специалистов, которые могут использовать эти знания при оптимизации своих алгоритмов.

Таким образом, изучение методов сортировки и их сравнение с помощью визуализации является важным шагом для повышения качества программирования и оптимизации алгоритмов.

# **Список источников.**

1. Асимптотика алгоритмов – <https://www.tutorialspoint.com/data_structures_algorithms/asymptotic_analysis.htm#:~:text=Asymptotic%20analysis%20of%20an%20algorithm,case%20scenario%20of%20an%20algorithm>, дата обращения 13.04.2022
2. Генри С. Уоррен – «Алгоритмические трюки для программистов», 2-е издание.
3. Дж. Брукшир, Д. Брилов, М. Вильямс – «Компьютерные науки. Базовый курс.» 13-е издание – 2019 – 992 с.
4. Дж. Макконелл – «Анализ алгоритмов. Вводный курс.», 2-е дополненное издание.
5. Микки Нардо. Учебное пособие по Tkinter для новичков – http://www.russianlutheran.org/python/nardo/nardo.html, дата обращения 13.04.2022
6. С. Дасгупта, Х. Пападимитриу, У. Вазирани – «Алгоритмы.»
7. Сравнение алгоритмов сортировки – <https://habr.com/ru/post/274017/>, дата обращения 13.04.2022
8. Т. Кормен – «Алгоритмы. Вводный курс.»
9. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн – «Алгоритмы. Построение и анализ.» 3-е издание.
10. Tkinter Python – <https://www.youtube.com/playlist?list=PLQAt0m1f9OHsd6U5okp1XLoYyQR0oBjMM>, дата обращения 13.04.2022
11. tkinter — Python интерфейс для Tcl/Tk | Python 3 – <https://digitology.tech/docs/python_3/library/tkinter.html>, дата обращения 13.04.2022