МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Допущен к защите

Заведующий кафедрой ПМИ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Разова

**Сравнительный анализ алгоритмов устойчивой сортировки**

Курсовой проект по дисциплине  
«Проектная и научно-исследовательская деятельность»

Выполнил студент группы ПМИб-2301-52-00  Жавнерко Максим Адреевич

Руководитель к.пед.н., доцент кафедры ПМИ  Котельникова Анастасия Валерьевна

Работа защищена с оценкой     \_\_\_\_.\_\_\_\_.2024 г.

Члены комиссии:     /     /

    /     /

Киров 2024

**Содержание**

[Введение 3](#_Toc167113955)

[1. Теоретические основы алгоритмов сортировки 5](#_Toc167113956)

[1.1 Постановка задачи 5](#_Toc167113957)

[1.2 Сортировка вставками 6](#_Toc167113958)

[1.3 Сортировка пузырьком 8](#_Toc167113959)

[1.4 Сортировка слиянием 9](#_Toc167113960)

[1.5 Сортировка подсчетом 11](#_Toc167113961)

[1.6 Выводы по теоретической главе 13](#_Toc167113962)

[2. Реализация и сравнение алгоритмов сортировки 14](#_Toc167113963)

[2.1 Сортировка вставками и пузырек 14](#_Toc167113964)

[2.2 Сортировка слиянием 15](#_Toc167113965)

[2.3 Сортировка подсчетом 15](#_Toc167113966)

[2.4 Сравнение алгоритмов сортировки 17](#_Toc167113967)

[2.5 Вывод по практической главе 18](#_Toc167113968)

[Заключение 19](#_Toc167113969)

[Библиографический список 20](#_Toc167113970)

[Приложения 21](#_Toc167113971)

[Приложение 1 21](#_Toc167113972)

[Приложение 2 21](#_Toc167113973)

Введение

Один из первых прототипов современного алгоритма сортировки применялся в конце 19 века для ускорения обработки переписи населения. В США американец Герман Холлерит создал сортировальную машину, работа которой основывалась на методах поразрядной сортировки.

В дальнейшем история алгоритмов связана с развитием [ЭВМ](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%BE-%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0), считается, что программа сортировки стала первой программой для ЭВМ. Некоторые конструкторы называли задачу сортировки данных наиболее характерной нечисловой задачей для вычислительных машин. В 1945 году [Джон фон Нейман](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D0%BC%D0%B0%D0%BD,_%D0%94%D0%B6%D0%BE%D0%BD_%D1%84%D0%BE%D0%BD) для тестирования ряда команд разработал программы [сортировки методом слияния](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D0%B5%D0%BC). В том же году немецкий инженер [Конрад Цузе](https://ru.wikipedia.org/wiki/%D0%A6%D1%83%D0%B7%D0%B5,_%D0%9A%D0%BE%D0%BD%D1%80%D0%B0%D0%B4) разработал программу для [сортировки методом простой вставки](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%B2%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0%D0%BC%D0%B8).

Объектом исследования являются алгоритмы сортировки данных, их слабые и сильные стороны.

Целью курсового проекта является формирование рекомендации по выбору алгоритма

Для достижения цели курсовой работы были поставлены следующие задачи:

1. Провести обзор научных исследований, посвященных алгоритмам стабильной сортировки. Изучить основные алгоритмы, их характеристики, преимущества и недостатки, а также проанализировать примеры их применения в различных областях.
2. Описание и анализ основных алгоритмов стабильной сортировки.
3. Найти или сгенерировать доступные наборы данных из различных предметных областей, обеспечивая разнообразие по сортируемым элементам.
4. Реализовать выбранные алгоритмы и применить их для сортировки на подготовленных наборах данных.
5. Сравнить эффективность различных алгоритмов на основе выбранных метрик качества.
6. Сформулировать выводы о применимости каждого алгоритма, а также дать рекомендации по выбору оптимального метода.

В первой главе проведен обзор существующих алгоритмов и их теоретическое время работы, а также описана их область применения. Вторая глава посвящена анализу производительности. Приложение содержит листинг программы.

1. Теоретические основы алгоритмов сортировки
   1. Постановка задачи

Пусть ­ входная последовательность элементов, где

Определим отношение порядка на элементах

Сортировка называется стабильной, если для любых двух индексов и где , выполняется следующее условие:

Если , то , где и .

Для эффективного алгоритма сортировки время выполнения стремится к в лучшем и среднем случае. Менее эффективные алгоритмы демонстрируют время выполнения в худшем случае. Идеальным сценарием является линейная временная сложность . Стоит отметить, что для алгоритмов сортировки, основанных на сравнении элементов, является нижней границей сложности, то есть не существует алгоритма сравнения, способного отсортировать массив быстрее, чем за [7, стр 364, 365].

Важно учитывать не только временную, но и пространственную сложность алгоритма. Многие алгоритмы сортировки требуют выделения дополнительной памяти, которая, как правило, составляет , то есть не зависит от размера входных данных. При анализе пространственной сложности не учитывается память, занимаемая исходным массивом, а также постоянные затраты, не зависящие от входных данных, такие как объем программного кода, что также соответствует .

Таким образом, при оценке эффективности алгоритма сортировки ключевыми параметрами являются временная сложность, определяющая скорость работы алгоритма, и пространственная сложность, характеризующая объем требуемой дополнительной памяти. Выбор оптимального алгоритма сортировки для конкретной задачи определяется балансом между этими двумя характеристиками с учетом особенностей обрабатываемых данных.

В рамках исследования были рассмотрены несколько алгоритмов, гарантирующие стабильность сортировки:

1. Сортировка вставками (Insertion Sort): Алгоритм, основанный на последовательном перемещении элементов на свои места путем сравнения с уже отсортированной частью массива.
2. Сортировка слиянием (Merge Sort): Алгоритм, основанный на принципе "разделяй и властвуй", рекурсивно разбивающий массив на подмассивы, сортирующий их и затем объединяющий в отсортированный массив.
3. Сортировка подсчетом (Counting Sort): Алгоритм, работающий за линейное время, подсчитывающий количество элементов с каждым значением и на основе этих данных строящий отсортированный массив.
4. Пузырьковая сортировка (Bubble Sort): Алгоритм, основанный на последовательном сравнении соседних элементов и их перестановке, если порядок нарушен, повторяя этот процесс до полной сортировки.
5. Сортировка поразрядно (Radix Sort): Сортирует по цифрам, начиная с младших разрядов.
6. Блочная сортировка (Bucket Sort): Распределяет элементы по "блокам" и сортирует внутри каждого блока.
7. Timsort: Гибридный алгоритм, использует сортировку вставками и слиянием, разработан для работы с реальными данными.

Для программной реализации были выбраны сортировки вставками, слияниями, подсчетом, пузырьком и TimSort.

Далее необходимо рассмотреть алгоритмы вышеназванных сортировок, показать их принцип работы на конкретных примерах.

* 1. Сортировка вставками

1. Инициализация:
   * Считаем, что подпоследовательность уже отсортирована.
2. Итерации:
   * Для каждого от до :
     + Пока и :

На каждой итерации внешнего цикла подпоследовательность $A[1..i]$ отсортирована.

Доказательство корректности:

* Базовый случай: При подпоследовательность тривиально отсортирована.
* Индуктивный шаг: Предположим, что для подпоследовательность отсортирована. На шаге алгоритм вставляет элемент на правильную позицию в уже отсортированной подпоследовательности , сравнивая его с элементами в обратном порядке и сдвигая большие элементы вправо. После завершения внутреннего цикла оказывается на правильной позиции, и подпоследовательность становится отсортированной.
* Завершение: По завершении цикла , вся последовательность отсортирована.

Временная сложность:

* Худший случай: (когда входная последовательность отсортирована в обратном порядке)
* Средний случай:
* Лучший случай: (когда входная последовательность уже отсортирована)

Пространственная сложность: (алгоритм сортирует на месте)

Стабильность: Сортировка вставками является стабильной, так как внутренний цикл перемещает элементы только в том случае, если они строго больше, сохраняя порядок равных элементов.

* 1. Сортировка пузырьком

1. Инициализация:
2. Итерации:
   * Пока :
     + Для каждого от 1 до :
       - Если :
         * Поменять местами и

На каждой итерации внешнего цикла наибольший неотсортированный элемент перемещается на свою окончательную позицию в конце последовательности.

Доказательство корректности:

* Базовый случай: Если , последовательность уже отсортирована.
* Индуктивный шаг: Предположим, что после итераций наибольших элементов находятся на своих окончательных позициях в конце последовательности. На -ой итерации алгоритм сравнивает соседние элементы и меняет их местами, если порядок нарушен. В результате наибольший из оставшихся неотсортированных элементов перемещается на -ю позицию, занимая своё окончательное место.
* Завершение: Цикл завершается, когда , что означает, что на последней итерации не было произведено ни одной перестановки. Это гарантирует, что все элементы находятся на своих окончательных позициях, и последовательность отсортирована.

Временная сложность:

* Худший случай: (когда входная последовательность отсортирована в обратном порядке)
* Средний случай:
* Лучший случай: (когда входная последовательность уже отсортирована)

Пространственная сложность: (алгоритм сортирует на месте)

Стабильность: Сортировка вставками является стабильной, так как внутренний цикл перемещает элементы только в том случае, если они строго больше, сохраняя порядок равных элементов.

* 1. Сортировка слиянием

Сортировка слиянием реализуется с помощью двух основных процедур:

1. Merge(L, R): Слияние двух отсортированных последовательностей и в одну отсортированную последовательность .
   * Инициализация: , , .
   * Пока и :
     + Если :
     + Иначе:
   * Скопировать оставшиеся элементы (если ) или (если ) в .
2. MergeSort(A): Рекурсивная сортировка последовательности .
   * Если длина() :
     + Вернуть
   * Иначе:
     + Вернуть

Доказательство корректности:

* Корректность Merge: Процедура Merge корректно сливает две отсортированные последовательности, так как на каждом шаге выбирается наименьший элемент из оставшихся в и и добавляется в , гарантируя возрастающий порядок элементов в .
* Корректность MergeSort:
  + Базовый случай: Если длина() , последовательность тривиально отсортирована.
  + Индуктивный шаг: Предполагаем, что корректно сортирует последовательности длины меньше . Для последовательности длины алгоритм разбивает ее на две подпоследовательности длины не более , рекурсивно сортирует их с помощью (что корректно по предположению) и затем сливает отсортированные подпоследовательности с помощью (что также корректно).
* Завершение: По завершении рекурсии возвращает отсортированную последовательность .

Временная сложность:

* Худший случай:
* Средний случай:
* Лучший случай:

Пространственная сложность: (inplace реализация)

Стабильность: Сортировка слиянием является стабильной, так как процедура при равных элементах и выбирает , сохраняя порядок элементов из исходной последовательности.

* 1. Сортировка подсчетом

Алгоритм:

1. Подсчет:
   * Создать массив размера , инициализированный нулями.
   * Для каждого элемента в :
2. Суммирование:
   * Для каждого от 1 до :
3. Распределение:
   * Создать массив размера .
   * Для каждого от до 1 (в обратном порядке):

Доказательство корректности:

* Подсчет: Массив хранит количество вхождений каждого значения из диапазона элементов .
* Суммирование: После суммирования содержит количество элементов в , меньших или равных .
* Распределение: Используя информацию из , алгоритм помещает каждый элемент на его окончательную позицию в , гарантируя сортировку.

Временная сложность:

Пространственная сложность: (для массивов и )

Стабильность: Сортировка подсчетом является стабильной, так как элементы с одинаковым значением обрабатываются в том же порядке, в котором они встречаются во входной последовательности, благодаря обратному порядку обработки в шаге "Распределение".

* 1. TimSort

Алгоритм:

1. Разбиение на прогоны:
   * Входной вектор делится на подмассивы фиксированного размера RUN.
   * Каждый подмассив сортируется с использованием insertionSortTim.
2. Слияние прогонов:
   * Последовательно сливаются пары соседних отсортированных прогонов (mergeTim), размер которых удваивается на каждой итерации (size).
   * Процесс слияния продолжается до тех пор, пока не будет получен единый отсортированный вектор.

Доказательство корректности:

* insertionSortTim: Гарантирует сортировку каждого прогона, так как использует классический алгоритм сортировки вставками.
* mergeTim: Обеспечивает корректное слияние отсортированных подмассивов, сохраняя относительный порядок элементов.

Временная сложность:

* insertionSortTim: в худшем случае, но эффективна для небольших массивов (прогонов).
* mergeTim: для слияния двух подмассивов размера n.
* Общее время: В среднем и худшем случае Timsort имеет сложность , как и другие эффективные алгоритмы сортировки. Благодаря использованию сортировки вставками для небольших прогонов, Timsort может показывать лучшую производительность на частично отсортированных данных.

Пространственная сложность:

* mergeTim: для временных векторов left и right.
* Общее пространство: в худшем случае.

Стабильность:

* insertionSortTim: Стабилен, так как не меняет порядок элементов с одинаковыми ключами.
* mergeTim: Также стабилен, так как сохраняет относительный порядок элементов при слиянии.
  1. Выводы по теоретической главе

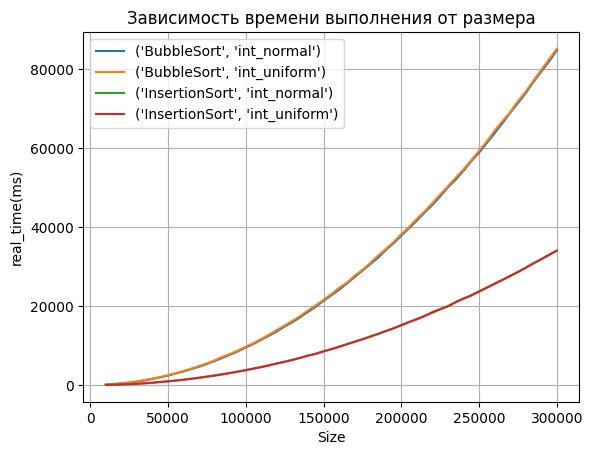
Сортировка вставками и пузырьковая сортировка, обладая временной сложностью в общем случае, демонстрируют низкую эффективность при работе с большими объемами данных. Однако, их простота реализации и минимальные требования к дополнительной памяти () делают их подходящим выбором для сортировки небольших массивов.

Сортировка слиянием отличается стабильной временной сложностью , гарантируя предсказуемое время работы независимо от характера входных данных.

Сортировка подсчётом является наиболее эффективным алгоритмом из рассмотренных, демонстрируя линейную временную сложность , где k - диапазон значений сортируемых элементов. Однако, эффективность этого алгоритма напрямую зависит от диапазона сортируемых значений: при большом значении k потребление памяти может стать неприемлемым.

1. Реализация и сравнение алгоритмов сортировки
   1. Сортировка вставками и пузырек

Проанализируем средний случай:

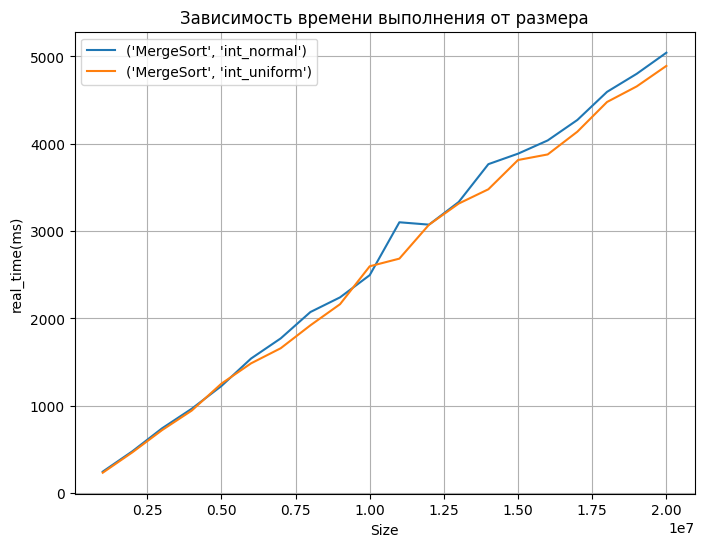


Нижние 2 кривые -­ это сортировка вставками. Верхние ­ сортировка пузырьком. Так как кривые зависимости, соответствующие нормальному и равномерному распределению, почти слиплись, то можно сказать, что эти сортировки не чувствительны к начальному распределению данных.

График иллюстрирует правильность временной сложности, рассчитанной теоретически.

Подробные результаты бенчмарков представлены в таблицах в приложении (ЗДЕСЬ БУДЕТ ССЫЛКА НА ПРИЛОЖЕНИЕ)

* 1. Сортировка слиянием

Проанализируем средний случай:

Можно подумать, что зависимость линейная, но на самом деле это не так. Дело в том, что на больших размерах входных данных сложность nlogn очень близка линейной сложности.

Прямые, соответствующие разным распределениям, почти совпадают, поэтому можно говорить, что алгоритм не чувствителен к распределению данных.

* 1. Сортировка подсчетом

Проанализируем лучший случай (мало различных значений):

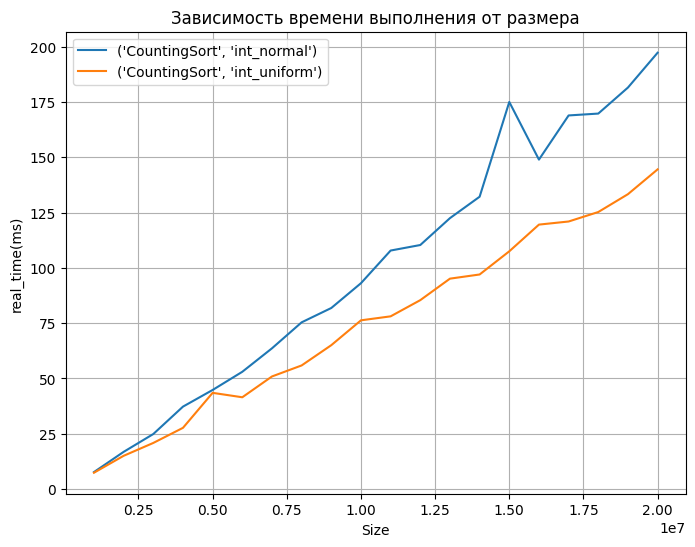
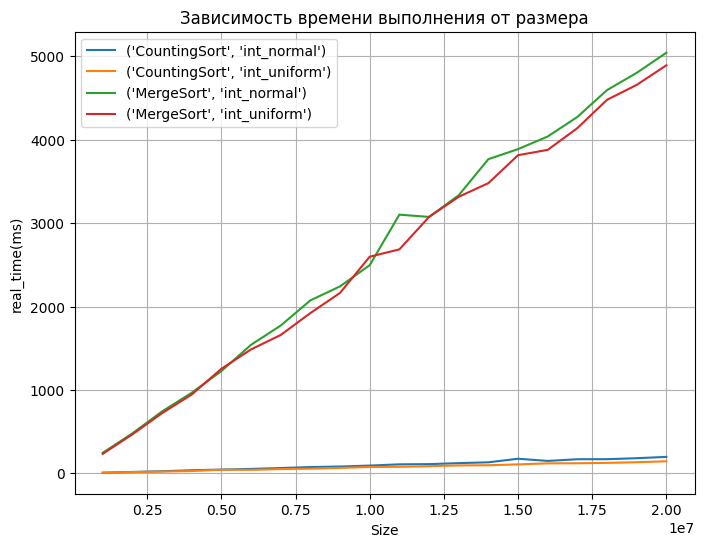


График иллюстрирует четкую линейную зависимость от размера входных данных, что подтверждается теоретически.

Разница между распределениями есть. Дело в том, что в нормальном распределении больше различных чисел.

Хорошо видна разница, на сколько быстрее работает сортировка подсчетом на оптимальных данных в отличие от сортировки слиянием:



Полное сравнение в таблицах находится в приложении (ЗДЕСЬ БУДЕТ ССЫЛКА НА ПРИЛОЖЕНИЕ)

2.4 Сравнение алгоритмов сортировки

Сравнительный анализ заключается в том, чтобы сравнивать реальное время, за которое выполняется алгоритм.

Для разработки и анализа методов сортировки мы будем использовать язык программирования C++20, библиотека Google Benchmark (<https://github.com/google/benchmark> ). Все алгоритмы были реализованы в среде разработки Visual Studio 2022 (система сборки стандартная MSBuild).

Google Benchmark сама определяет количество итераций до стабилизации результата для каждого теста.

2.5 Вывод по практической главе

Заключение

Библиографический список

1. [Дональд Кнут](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%B4_%D0%9A%D0%BD%D1%83%D1%82" \o "Дональд Кнут)*.* Искусство программирования, том 3. Сортировка и поиск = The Art of Computer Programming, vol.3. Sorting and Searching. — 2-е изд. — М.: [«Вильямс»](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2007. — С. 83—95. — [ISBN 5-8459-0082-4](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%B3/5845900824).
2. [Дональд Кнут](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%B4_%D0%9A%D0%BD%D1%83%D1%82)*.* Искусство программирования, том 3. Сортировка и поиск = The Art of Computer Programming, vol.3. Sorting and Searching. — 2-е изд. — М.: [«Вильямс»](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2007. — С. 192—201. — [ISBN 5-8459-0082-4](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%B3/5845900824).
3. Ефимов С. С. Обзор методов распараллеливания алгоритмов решения некоторых задач вычислительной дискретной математики // МСиМ. 2007. №1 (17). URL: https://cyberleninka.ru/article/n/obzor-metodov-rasparallelivaniya-algoritmov-resheniya-nekotoryh-zadach-vychislitelnoy-diskretnoy-matematiki (дата обращения: 15.12.2022).
4. Список рассылки разработчиков Python : сайт. – URL: https://mail.python.org/pipermail/python-dev/2002-July/026837.html (дата обращения: 17.12.2022)
5. Nicolas Auger, Vincent Jugé, Cyril Nicaud, Carine Pivoteau. On the Worst-Case Complexity of TimSort. 26th Annual European Symposium on Algorithms (ESA 2018), Aug 2018, Helsinki, Finland. pp.4:1–4:13, 10.4230/LIPIcs.ESA.2018.4. hal-01798381
6. Поразрядная сортировка: сайт. – URL: http://algolist.ru/sort/radix\_sort.php (дата обращения: 07.01.2023)
7. Mishra, Aditya Dev, and Deepak Garg. "Selection of best sorting algorithm." *International Journal of intelligent information Processing* 2.2 (2008): 363-368.

Приложения

Приложение 1

ТУТ БУДЕТ КОД

Приложение 2

ТУТ БУДЕТ КОД