Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

СЕМЕСТРОВАЯ РАБОТА

по дисциплине «Алгоритмы и анализ сложности»  
«Экспериментальный анализ различных методов сортировки»

Обучающийся: Валирахманов Данис Насимович гр. 09–232

(ФИО студента) (Группа)

Руководитель: к.ф.-м.н., доцент КСАИТ, А. В. Васильев

Казань – 2024

**Оглавление**

[Введение 3](#_Toc162345006)

[Постановка задачи 4](#_Toc162345007)

[Методика проведения эксперимента 5](#_Toc162345008)

[Полученные результаты 6](#_Toc162345009)

[Заключение 7](#_Toc162345010)

[Приложение 1. Программный код 8](#_Toc162345011)

Введение

Будучи школьником я часто сталкивался с алгоритмическими задачами ,которые требовали использование сортировок. Часто именно от их эффективности зависит скорость выполнения конечной программы. Я часто задавался вопросом ,что скрывается под стандартной сортировкой языка и является ли она самой лучшей ,что если есть возможности найти более эффективный вариант. В своей работе я реализовал и сравнил более 11 сортировок и изложил свои результаты в подробных графиках и таблицах.

Постановка задачи

1**. Изучение алгоритмических задач, требующих применения сортировок, с целью повышения эффективности выполнения программ.**

**2. Рассмотрение стандартной сортировки в используемом языке программирования и поиск более эффективных вариантов.**

**3. Проведение сравнительного анализа более чем 11 различных сортировок с целью определения их производительности и эффективности.**

**4. Реализация и исследование результатов работы сортировок с последующим описанием полученных данных в виде графиков и таблиц.**

**5. Оценка сложности и временной эффективности каждой из протестированных сортировок на различных объемах данных.**

**6. Идентификация наиболее подходящих сортировок для конкретных типов данных и задач с целью оптимизации процессов сортировки в программах.**

Методика проведения эксперимента

Сравнение будет проходить на базе ноутбука ASUS Vivobook:

Ноутбук на ОС Windows 10 ,ЦП intel core I5 10300H ,видеокарта GTX 1650 ,ОП 8 гб 1 плашка. Проект написан на C# ,версия .Net 6, используется IDE : VS.

Будем сравнивать 11 различных сортировок ,в том числе и на встроенной в язык сортировке. Стоит отметить ,что встроенная сортировка в C# написана на С++ ,а также использует различные оптимизации памяти, поэтому в сравнении будет использоваться как эталон времени. Сортировки будут проверяться со включенным сборщиком мусора ,поэтому ожидается ,что некоторые результаты будут отличаться от ожидаемых.

Сравнение будет проводится на 5 массивах различной длины: {50, 500 5000, 50000, 500000}, на 4 различных типах данных: int ,byte ,string ,Date (тип Date написан был мною лично ,лишь позже я узнал ,что в C# есть встроенный тип Дата, сравнение было сделано при помощи наследования интерфейса IComparable ,что позволило моему классу работать со встроенной в язык сортировкой) ,а также на 3 различных типах массива: массив случайных чисел ,массив частично отсортированных чисел ,массив с большим количеством одинаковых чисел. В общем случае нам нужно рассмотреть более 600 случаев(5\*4\*3\*10).

Про генерацию данных:

Каждую из сортировок я тестировал более 10 раз ,каждый раз генерируя новый массив. Для этого я создал класс DataGenerator ,в котором реализовал методы для создания массивов 3 различных видов. Случайный массив я создавал с помощью встроенной библиотеки Random, частично отсортированный массив создавался по принципу: сначала создаю массив случайных чисел ,потом сортирую его при помощи встроенный в язык сортировки ,после чего создаю лист длинной 10% от длинны изначального массива уникальных значений индексов ,каждый элемент массива с индексом из этого листа заменяется случайным значением. Таким образом мы получаем частично отсортированный массив с гарантией того ,что 10% из этого массива это случайные числа ,а остальные элементы отсортированные. Подобным образом создаем массив с большим количеством одинаковых элементов, изначально создавая случайный элемент и заполняя им весь массив ,а затем внедряя вместо 10% случайные значения.

Благодаря большому количеству элементов ,а также многократным тестам для каждой сортировки мы будем получать схожие по характеристикам массивы ,благодаря закону Больших чисел. Если бы мы все сравнивали на одном конкретном массиве ,то столкнулись бы с проблемой ,что не существует идеально среднего массива ,соответственно сравнение исключительно на этом массиве было бы корректно ,так как он мог оказаться более хорошим случаем для одной сортировки и более плохим для другой сортировки ,и данные получились бы необъективными. Поэтому я выбрал подход генерации каждый раз нового массива!

Разброс данных:

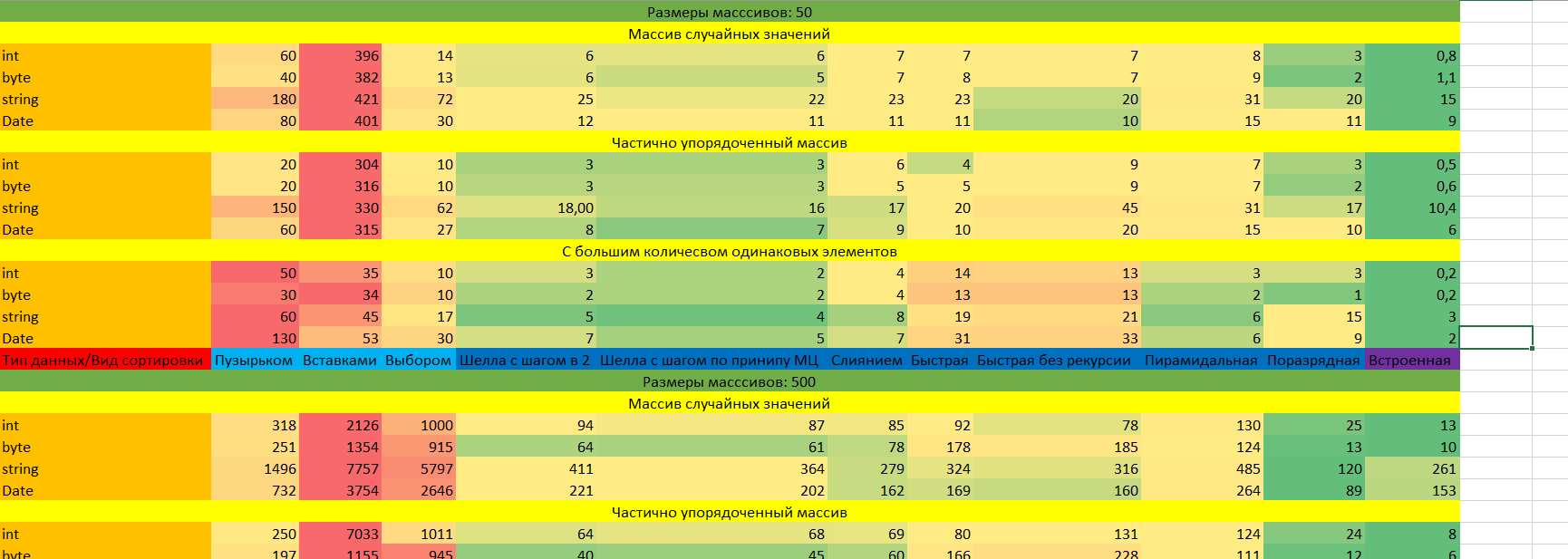
1. Возможные значения для типа int составляют диапазон от 0 до 99.
2. Тип byte как и требуется по заданию составляет диапазон от 0 до 9.
3. Тип string это строки случайной длины от 5 до 10 ,со случайным значением строки (заглавные и строчные буквы ,без цифр)
4. Тип Date это данные в виде ДД.ММ.ГГ. Дни варьируются от 0 до 29 (чтобы не думать о проблеме февраля),месяцы от 1 до 12, года от 0 до 9 включительно (чтобы программе чаще приходилось сравнивать не только по годам ,но и по дням).

Все данные собраны в миллисекундах!

Полученные результаты

Так как разброс данных достаточно весомый ,данных много и они распределяются по группам ,то было решено ,что лучший способ визуализации это таблица подобного типа (гистограммы и т.д. плохо подходят в подобном случае), где я распределил цветом наилучшее время (зеленый) и наихудшее(красный).

P.S позже я узнал что так нельзя было делать ,но так как в задании не было четко указано что таблицы это плохо ,я решил обратиться в интернет с запросом: “ какой график применить ,если для данных чем меньше тем лучше”. На данный вопрос был представлен ответ только на данном сайте <https://ux.stackexchange.com/questions/122635/plotting-a-comparison-graph-where-less-is-better> ,где человек столкнулся с точно такой же проблемой. Ему порекомендовали использовать таблицы. Я был согласен с мнением отвечающего ,а также открыл возможность распределения значений по цветам. Лично я считаю ,что этот способ визуализации куда нагляднее нежели график ,так как позволяет охватить картину целиком, а также обладает четким показателем того во сколько раз быстрее одна сортировка нежели другая. Наш глаз очень хорошо различает оттенки ,поэтому ему без труда удается представлять как будет выглядеть цвет в палитре от красного до зеленого в зависимости от значения. Прошу понять мою ситуацию! :



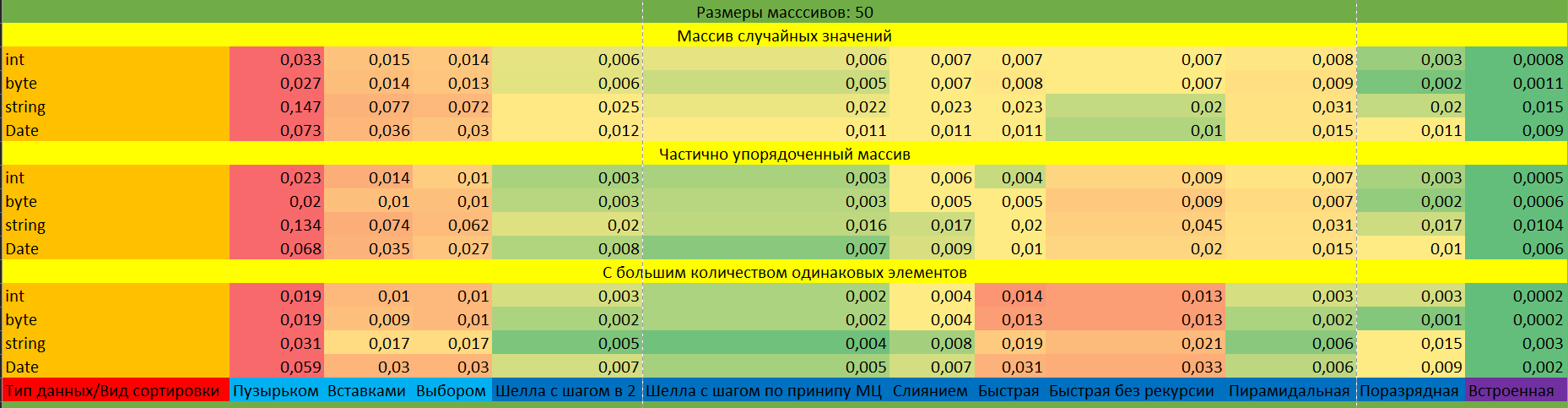
Список реализованных сортировок:

Медленные:

1. Пузырек
2. Вставками
3. Выбором

Быстрые:

1. Сортировка Шелла с шагом /2
2. Сортировка Шелла с шагом по последовательности Марцина Циура
3. Слиянием
4. Быстрая с использованием рекурсии
5. Быстрая без рекурсии
6. Пирамидальная
7. Поразрядная
8. Встроенная

Сравнение сортировок на массивах с длинной : 50

Как мы видим лучше всех справилась ,как и ожидалось ,встроенная сортировка, хуже всех себя показала сортировка Пузырьком, она сильно проигрывает даже на фоне других медленных сортировок. Из быстрых сортировок лучшее время показали: Поразрядная и Шелла с шагом по принципу МЦ. Шелла с шагом по принципу МЦ справилась лучше ,чем стандартная сортировка Шелла, данный результат является ожидаемым ,т.к эмпирическая последовательность МЦ доказано является лучшим выбором шагов для сортировки Шелла с массивами до 4000+ элементов. Также стоит отметить ,что она отлично себя показала на массивах с большим количеством одинаковых элементов.

 Сравнение сортировок на массивах с длинной : 500

В данной таблице можно заметить интересный момент: “эталонные” результаты встроенной сортировки иногда проигрывают Поразрядной сортировке. Данное явление мы можем наблюдать при сортировке ссылочных типов данных: Date и String. Но на массивах с большим количеством одинаковых элементов Поразрядная сортировка полностью проигрывает встроенной. Интересно также ,что сортировка Пузырьком показала время намного лучшее ,нежели чем у остальных медленных сортировок ,хотя на предыдущих массивах она сильно отставала. Уже на таких маленьких размерах массивов мы можем заметить главную проблему быстрой сортировки: она очень плохо работает с массивами с большим количеством одинаковых элементов (ее сложность в худшем случае N^2) ,ее время приближается к самым худшим результатам данной таблицы.

Сравнение сортировок на массивах с длинной : 5000

Мы опять можем наблюдать феномен из предыдущий таблицы ,Поразрядная сортировка в некоторых ситуациях оказывается быстрее! Четко прослеживается проблема Быстрой сортировки, а сортировка Пузырьком вновь сильно отстает от других медленных сортировок. Стоит отметить ,хоть размер массива уже больше 4000 ,но сортировка Шелла с шагом по принципу МЦ все еще лучше стандартной сортировки Шелла! Также можно заметить ,что Поразрядная сортировка в среднем быстрее других быстрых сортировок.

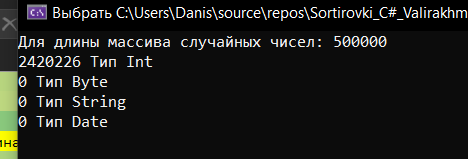
Сортировки Вставками и Выбором показывают в среднем одинаковое время.

 Сравнение сортировок на массивах с длинной : 50000

Феномен Поразрядной сортировки сохраняется и на массивах с 50000 элементами. Сортировка Пузырьком стабильно показывает плохие результаты. На данный момент уже можно наблюдать проблемы рекурсивной реализации Быстрой сортировки ,программе стало не хватать памяти и она завершается с ошибкой stack overflow. Проблема связана с тем ,что рекурсия стала слишком глубокой. Интересно ,что в среднем нет разницы в скорости рекурсивной и без рекурсивной реализации Быстрой сортировки для массива случайных чисел ,но уже для массивов частично упорядоченных и с большим количеством одинаковых чисел рекурсивная сортировка показывает лучшие результаты! Среди быстрых сортировок виден только один четкий победитель, остальные ведут себя неоднозначно ,иногда одни быстрее ,иногда другие.

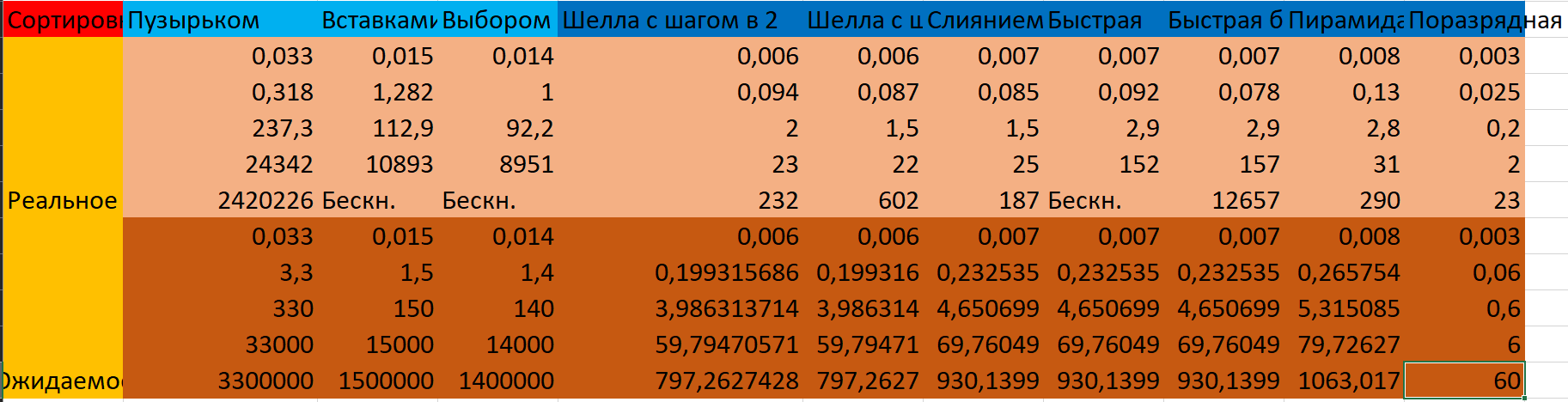
 Сравнение сортировок на массивах с длинной : 500000

Примечание: медленные сортировки не были протестированы на полмиллиона элементах связи с очень большим временем ожидания ,тем не менее ради испытания один результат был собран для типа int для сортировки Пузырьком. Результат был удручающим, порядка 40 минут.

(запускал только для массива типа int[]) Считаю ,что мы можем опираться в данном случае на результаты предыдущих данных ,так как наблюдали четкое поведение медленных сортировок.

Заметим ,что Поразрядная сортировка уже имеет меньший отрыв от встроенной ,возможно на большем количестве элементов отрыв и вовсе пропадет. Тем не менее Поразрядная сортировка все еще является лучшей среди быстрых сортировок! Быстрая сортировка с рекурсией вовсе перестала работать ,а быстрая без рекурсии показывает очень плохое время. Сортировка Шелла по МЦ стала сильно проигрывать стандартной Шелла ,что доказывает ,что последовательность лучше всего работает до 4000 элементов. Если же рассматривать сортировки после Поразрядной ,то также себя неплохо показывает сортировка Слиянием. После нее идет Пирамидальная ,затем сортировки Шелла ,после чего Быстрые сортировки.

Сравнение результатов с ожидаемым значением:

Хотелось бы так же сравнить некоторые сортировки с их ожидаемым временем. Для этого будем сравнивать среднюю сложность сортировок на массивах от 500 до 5000 элементов с шагом в 500 элементов, для типа int, на массивах со случайными элементами. Сравнивать будем не все сортировки ,а лишь наиболее необычные или типовые случаи, хотя небольшую таблицу все же представлю. 

(в этой таблице размеры массивов от 50 до 500000)

1. Сортировка Пузырьком:

Ее я решил взять для сравнения как типового представителя медленных сортировок. Ее средняя сложность это O(n^2):

Мы можем наблюдать ,что графики очень схожи ,реальное время лишь немногим оказывается быстрее ожидаемого. Это можно объяснить тем ,что для данной сортировки массив с типом данных от 0 до 99 случай чуть лучше среднего. Получаем ,что наша сортировка реализовано верно, ее сложность соответствует теоретической.

1. Сортировка Шелла:

Поведение стандартной сортировки Шелла мне показалось очень странной ,ее результаты были намного лучше ожидаемых ,даже если я брал лучшее для нее сложность O(n \* log(n)) ,другие же сложности дают еще более отличающийся вариант

С чем связано подобное отклонение сказать трудно ,возможно дело в оптимизации c# и удачным видом данных для данной сортировки ,ведь значения int лежат всего лишь от 0 до 99. В целом можно сказать ,что графики ведут себя похожим образом ,а значит эта сортировка соответствует своей сложности.

1. Сортировка Поразрядная:

Также хотелось бы рассмотреть данную сортировку ,так как она имеет отличную от остальных среднюю сложность O(nK) ,где K – максимальная длина элемента.

Как мы видим ,во всех 3 случаях реальное время оказывается меньше ,но графики при этом схожи по поведению. Я думаю ,что это связано с типом данных на котором мы рассматривали данные сортировки, а также с оптимизацией самого c#.

Причины результатов:

Хотелось бы порассуждать о причинах некоторых результатов ,а также о нерассмотренных сторонах сортировок.

1. Поразрядная сортировка:

А) Пока я разрабатывал код программы я ни раз столкнулся с главной проблемой Поразрядной сортировки – она жестко зависит от типа данных. Свою программу я разрабатывал с использованием generic, это существенно уменьшало время на расширение типов данных и сокращало количество написанного кода. Для поразрядной же сортировки подобный трюк не сработает. Пришлось писать реализацию для всех 4 типов данных.

Б) Хоть в моих тестах этого не видно ,но у поразрядной сортировки могут быть достаточно большие требования по памяти.

В) Исходя из сложности данной сортировки она будет плохо работать на данных с большой длинной элементов ,но так как все значения не превышали в длину 4 ,то данный минус не всплыл.

2) Быстрая сортировка оказалось одной из самых отстающих в моих тестах ,но у нее есть ряд и достоинств:

A) Одним из основных достоинств быстрой сортировки является высокая скорость работы в среднем и лучшем случае. Сложность алгоритма быстрой сортировки в среднем составляет O(n\*log(n)), что делает его одним из самых эффективных алгоритмов сортировки.

Б) Быстрая сортировка позволяет проводить сортировку "на месте" (in-place sorting), что означает, что она не требует дополнительной памяти для работы с данными, кроме используемой для хранения самих данных.

В) Этот алгоритм эффективно справляется с большими объемами данных и хорошо работает на случайных или частично упорядоченных данным.

Г) Быстрая сортировка является рекурсивным алгоритмом, что делает ее компактной и легко читаемой в реализации.

Д) При правильной настройке параметров (например, выбором оптимального опорного элемента) быстрая сортировка может дать отличные результаты и обеспечить высокую производительность.

Также хотел бы заметить ,что быстрая рекурсивная сортировка чувствительна к размеру массива ,в отличие от остальных сортировок ,поэтому без рекурсивная сортировка является более универсальным вариантом ,но при этом проигрывает по скорости на массивах меньшего размера.

3) Сортировки Шелла:

А) Сортировки Шелла являются переходными сортировками между быстрыми и медленными в зависимости от выбора шага.

Б) Как показала практика на маленьких элементах лучше себя показывает последовательность Марцина Циура, но на больших выигрывает стандартная. Поэтому можно сделать вывод ,что комбинирование последовательностей даст наилучший результат.

**Заключение**  
1. Поразрядная сортировка демонстрирует хорошие результаты на массивах различной длины, особенно на массивах с большим количеством одинаковых элементов.

2. Сортировка Шелла с шагом по принципу МЦ может быть эффективнее стандартной Сортировки Шелла на массивах до 4000 элементов.

3. Быстрые сортировки могут иметь проблемы с производительностью на массивах с большим количеством одинаковых элементов.

4. Сортировка Пузырьком, хотя обычно считается одной из самых медленных, в определенных случаях может показать лучшие результаты.

5. Сортировка слиянием и пирамидальная сортировка также могут показывать хорошие результаты после поразрядной сортировки.

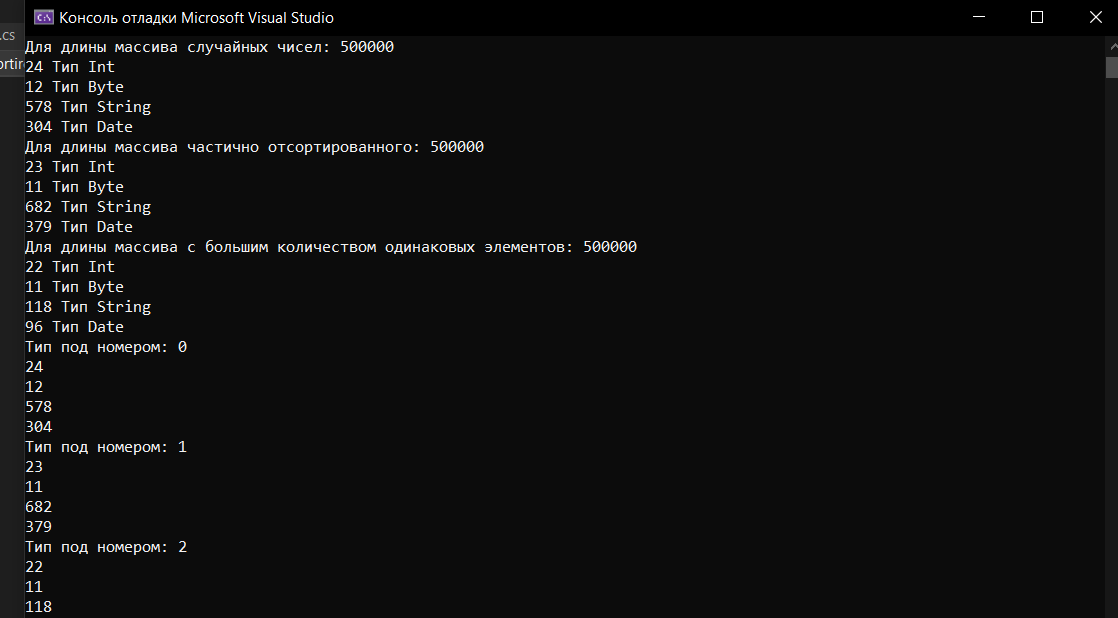
6. Встроенная сортировка является наилучшим вариантом для сортировки в большинстве случаев, однако поразрядная сортировка является лучшей среди быстрых сортировок.  
7. Проведенные эксперименты показывают, что эффективность различных алгоритмов сортировки зависит не только от объема данных, но и от структуры самих данных. Не существует универсального алгоритма сортировки, который будет идеально подходить для всех сценариев.

8. Выбор оптимального алгоритма сортировки требует анализа и понимания особенностей конкретной задачи, включая тип данных, их распределение, а также требования к производительности. Необходимо учитывать эти факторы при выборе алгоритма для конкретной задачи.

Приложение 1. Программный код

Код написан на языке C# ,на Vхisual Studio ,версия .Net 6. Было реализовано 4 класса:

1. Program: в нем я сделал удобный вывод и выбор размера тестируемого массива, пространство для тестов.
2. DataGenerator: класс реализует generic создание массивов различного типа ,размеров и типов заполнения массива.
3. Sortirovki: класс реализует все сортировки из моей лабораторной работы, методы являются статическими, также использующими generic.
4. Date: класс реализующий тип Date ,содержит методы для удобного создания и вывода в консоль значений. Класс наследуется от интерфейса IComparable для использования в сортировках работающих с generic.

Пример вывода: 

Код опубликован в моем GitHub: <https://github.com/DV-program/SortProject.git>

Также в нем будет опубликован Excel файл с результатами тестов.